

Коллоквиум письменный, 2 вопроса, час времени.

- Построение градиентного спуска с помощью ряда Тейлора.

## Многомерный случай

Задача:

$$f(x) \rightarrow \min_x$$

Разложим в ряд Тейлора до первой производной

$$f(x) \approx f(x_0) + (x - x_0)^T \nabla f(x_0)$$

Когда достигается максимум  $\langle x - x_0, \nabla f(x_0) \rangle$ ?

Ответ: когда  $(x - x_0)$  и  $\nabla f(x_0)$  параллельны

Наибольшее возрастание:  $x - x_0 = \eta \nabla f(x_0)$

Наибольшее убывание:  $x - x_0 = -\eta \nabla f(x_0)$

- Стохастический градиентный спуск. Версия с инерцией.

## Стохастический градиент

Разложим функцию потерь:

$$L(D, \mu) = \frac{1}{N} \sum_{i=1}^N l(x_i, \mu).$$

$$\nabla_\mu L(D, \mu) = \frac{1}{N} \sum_{i=1}^N \nabla_\mu l(x_i, \mu).$$

Пусть  $I = (i_1, i_2, \dots, i_m)$  — небольшая подвыборка  $D$

Приблизим градиент:

$$\nabla_\mu L(D, \mu) \approx \frac{1}{m} \sum_I \nabla_\mu l(x_i, \mu).$$

## Стохастический градиентный спуск (SGD)

$$\sum_{i=1}^N f_i(x) \rightarrow \min_x$$

$\eta$  — величина шага,  $m$  — размер подвыборки

### ❶ Инициализация

$k = 0, x_k = \text{начальное приближение}$

### ❷ Шаг *примерно* в сторону сильнейшего убывания

$I := \text{случайная подвыборка размера } m$

$$x_{k+1} = x_k - \eta \sum_I \nabla f_i(x_k)$$

### ❸ Повторение до сходимости

$k := k + 1, \text{ перейти к 2}$

## SGD mini-batches

На каждой итерации:

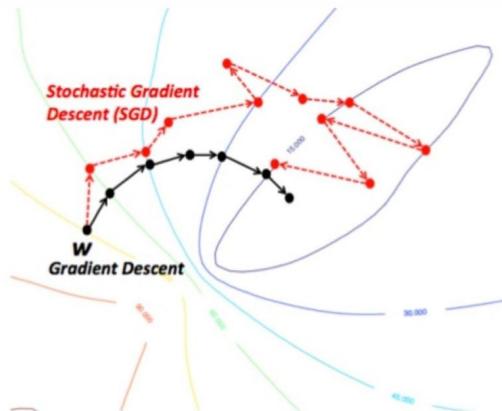
- Перемешать выборку
- Разбить выборку на равные части размера  $m$  (мини-батчи)

$$I_1 + I_2 + \dots + I_{N/m} = \{1, \dots, N\}$$

- Для  $j = 1, \dots, N/m$

$$x := x - \eta \sum_{I_j} \nabla f_i(x)$$

## SGD: шаги



Шум в оценке градиента помогает выпрыгивать из локальных оптимумов

## (S)GD + инерция (momentum)

$$f(x) \rightarrow \min_x$$

$\eta$  — величина шага,  $\alpha$  — влияние инерции

### ① Инициализация

$$k = 0, \quad x_k = \text{начальное приближение}, \quad m_k = 0$$

### ② Шаг в сторону сильнейшего убывания

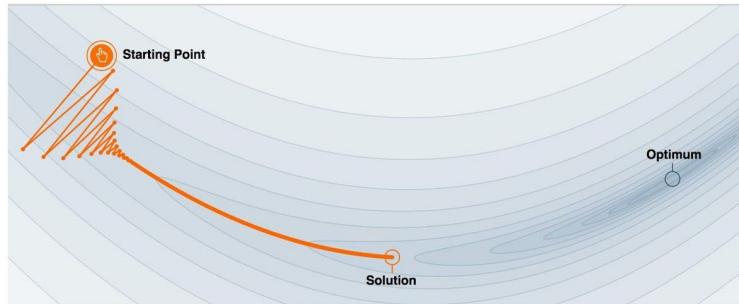
$$m_{k+1} := -\eta \nabla f(x_k) + \alpha m_k$$

$$x_{k+1} = x_k + m_{k+1}$$

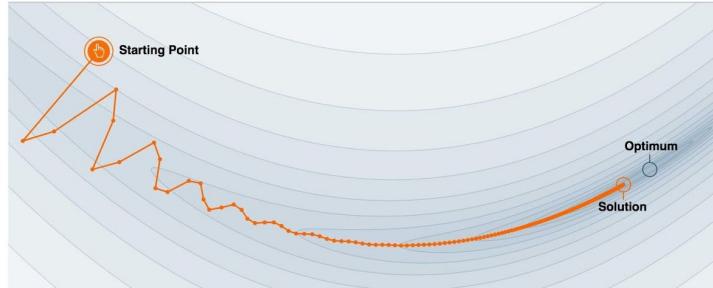
### ③ Повторение до сходимости

$$k := k + 1, \quad \text{перейти к 2}$$

## Сходимость без инерции



## Сходимость с инерцией



«Тяжелый мячик катится по поверхности»

## (S)GD + переменный шаг

$$f(x) \rightarrow \min_x$$

$\eta$  — базовая величина шага (гиперпараметр)

### ① Инициализация

$k = 0, \quad x_k = \text{начальное приближение}$

### ② Шаг в сторону сильнейшего убывания

$$x_{k+1} = x_k - \frac{\eta}{k} \nabla f(x_k)$$

### ③ Повторение до сходимости

$k := k + 1, \quad \text{перейти к 2}$

3. Построение метода Ньютона с помощью ряда Тейлора.

## Метод Ньютона

$$f(x) \rightarrow \min_x$$

$\eta$  — величина шага (гиперпараметр)

### ① Инициализация

$k = 0, \quad x_k = \text{начальное приближение}$

### ② Шаг в сторону примерного минимума

$$x_{k+1} = x_k - \eta (\nabla^2 f(x_k))^{-1} \nabla f(x_k)$$

### ③ Повторение до сходимости

$k := k + 1, \quad \text{перейти к 2}$

## Метод Ньютона

### • Плюсы:

- Сходится за меньшее число шагов по сравнению с GD

### • Минусы:

- Долго вычислять Якобиан  $\nabla^2 f(x_k)$

На практике используют квази-Ньютоновские методы, такие как L-BFGS

4. Покоординатный спуск.

## Покоординатный спуск

$$f(x) = f(x_1, x_2, \dots, x_n) \rightarrow \min_x$$

### ❶ Инициализация

$k = 0, \quad x^k = \text{начальное приближение}$

### ❷ Минимизируем вдоль координаты $i$ для $i = 1, \dots, n$

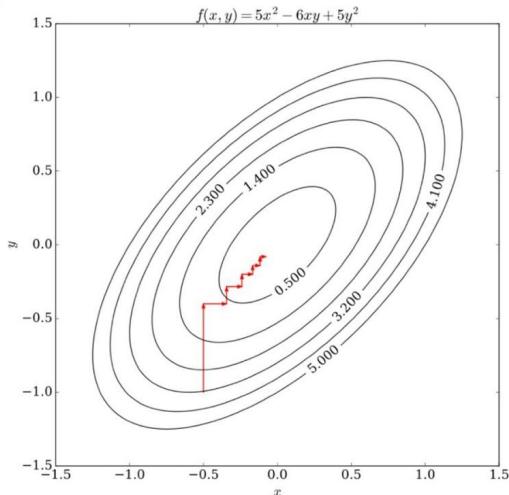
$$z^* = \arg \min_z f(x_1, \dots, x_{i-1}, z, x_{i+1}, \dots, x_n)$$

$$x^{k+1} = (x_1^k, \dots, x_{i-1}^k, z^*, x_{i+1}^k, \dots, x_n^k)$$

$$k := k + 1$$

### ❸ Повторение до сходимости

## Покоординатный спуск: шаги



## Покоординатный спуск

### • Плюсы:

- Не нужно вычислять производную

### • Минусы:

- Долго работает

5. Бустинг для задачи регрессии, связь с градиентом функции потерь.

## Бустинг для регрессии

$L = \{x_i, y_i\}_{i=1}^N$  — обучающая выборка

$K$  — количество деревьев,  $\lambda$  — скорость обучения

- Инициализация

$$a(x) := 0, \quad r_i = y_i \text{ для } i = 1, \dots, N$$

- Для  $k = 1, \dots, K$

- Новая обучающая выборка

$$L' = \{x_i, r_i\}_{i=1}^N$$

- Обучение решающего дерева  $a_k$  на  $L'$
- Обновление алгоритма и ошибки

$$a(x) := a(x) + \lambda a_k(x)$$

$$r_i := y_i - a(x_i) \quad \text{для } i = 1, \dots, N$$

## Остаток и градиент потерь

- Заметим, что остатки  $r_i$  могут быть найдены как антиградиент функции потерь по ответу модели  $a_i$ , посчитанный в точке ответа уже построенной композиции  $a_{N-1}(x_i)$ :

$$r_i = \boxed{y_i - a_{N-1}(x_i)} = -\frac{\partial}{\partial a_i} \frac{1}{2} \sum_{k=1}^l (a_k - y_k)^2 \Bigg|_{a_i = a_{N-1}(x_i)}$$

- То есть мы настраиваем следующее дерево на антиградиент потерь, тем самым мы делаем градиентный спуск в пространстве алгоритмов!

6. Бустинг для классификации. Алгоритм AdaBoost (без доказательства формулы для весов отдельных деревьев) и используемая верхняя оценка пороговой функции.

# Бустинг для классификации

$\{x_i, y_i\}_{i=1}^N$  — обучающая выборка

$$y_i \in \{-1, +1\}$$

$$a_i(x) \in \{-1, +1\} \text{ для } i = 1, \dots, K$$

$$a(x) = a_1(x) + \dots + a_K(x)$$

На итерации  $k$ :  $a_k(x) \sim \{x_i, r_i\}$

$$r_i = y_i - a(x_i) = y_i - (a_1(x_i) + \dots + a_{k-1}(x_i))$$

Добавляем компонент:

$$Q(a, a_{k+1}) = \sum_{i=1}^N [y_i \neq \text{sign}(a(x_i) + a_{k+1}(x_i))]$$

$$Q(a, a_{k+1}) \rightarrow \min_{a_{k+1}}$$

$$\sum_{i=1}^N [y_i \neq \text{sign}(f(x_i))] = \sum_{i=1}^N [y_i f(x_i) < 0] \quad \begin{array}{l} \text{Оптимизировать сложно!} \\ \text{Отступ на объекте} \end{array}$$

# Алгоритм AdaBoost

$L = \{x_i, y_i\}_{i=1}^N$  — обучающая выборка

- Инициализация решающей функции

$$f(x) := 0, \quad w_i = 1/N \text{ для } i = 1, \dots, N$$

- Для  $k = 1, \dots, K$

- Взвешенная обучающая выборка

$$L' = \{x_i, y_i, w_i\}_{i=1}^N$$

- Обучение решающего дерева  $t_k$  на  $L'$
- Вычисление  $\alpha_k$  по величине ошибки  $t_k$  на  $L'$
- Обновление алгоритма и весов

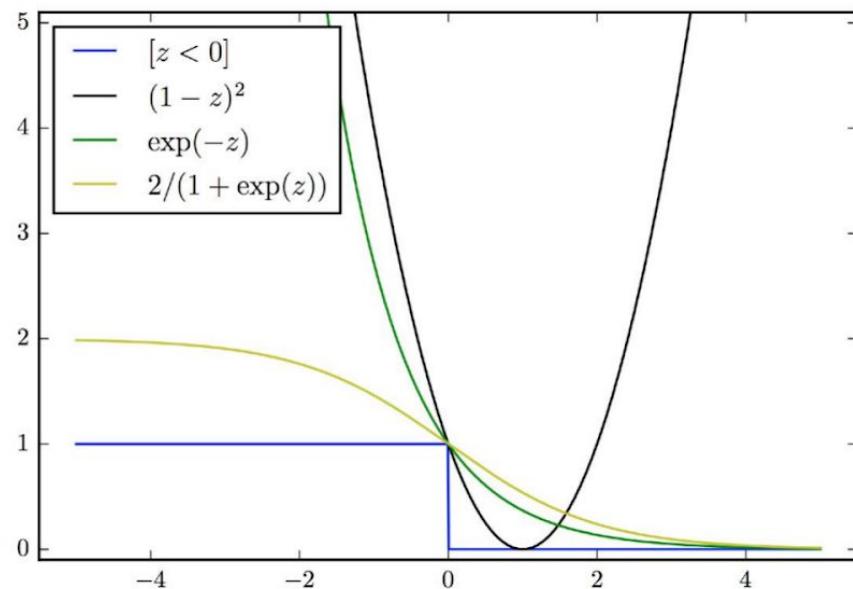
$$f(x) := f(x) + \alpha_k t_k(x)$$

$$w_i = \exp(-y_i f(x_i))$$

- Итоговый классификатор:  $a(x) = \text{sign}(f(x))$

## Верхние оценки

- $T(z) = \sum_{i=1}^N [z = y_i f(x_i) < 0]$



7. Градиентный бустинг для произвольной функции потерь.

## Функция потерь

- Ошибка на одном объекте:  $L(y, z)$
- MSE:  $L(y, z) = (y - z)^2$
- Логистическая функция потерь:  $L(y, z) = \log(1 + \exp(-yz))$

## Градиентный бустинг

1. Построить начальный алгоритм  $b_0(x)$

2. Для  $n = 1, \dots, N$ :

3. Вычислить сдвиги:

- $s = (-L'_z(y_1, a_{n-1}(x_1)), \dots, -L'_z(y_\ell, a_{n-1}(x_\ell)))$

4. Обучить новый базовый алгоритм:

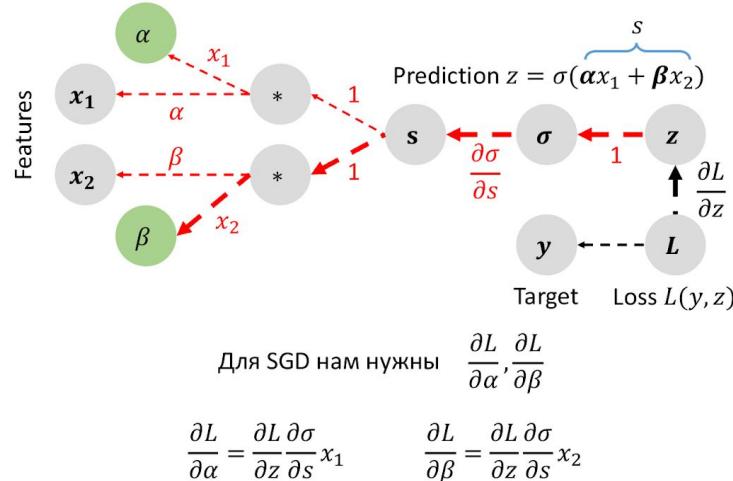
- $b_N(x) = \arg \min_b \frac{1}{\ell} \sum_{i=1}^{\ell} (b(x_i) - s_i)^2$

5. Добавить алгоритм в композицию:  $a_n(x) = \sum_{m=1}^n b_m(x)$

8. Нейронная сеть без скрытых слоев, вывод формулы производной функции потерь по одному из параметров.

Тут надо запомнить сам рисунок и выписать одну из нижних формул. Скрытого слоя нет так как после суммирования и нелинейного преобразования сразу считается loss

### Граф вычисления производных



9. Нейронная сеть с одним скрытым слоем, вывод формулы производной функции потерь по одному из параметров первого слоя. Идея алгоритма back-propagation.

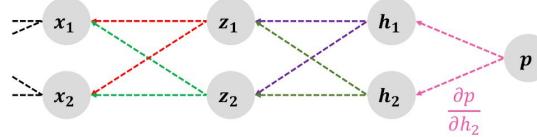
## MLP с 3 скрытыми слоями

На самом деле мы хотим менять параметры нейрона:

$$h_2 = \sigma(w_0 + w_1 z_1 + w_2 z_2)$$

Градиент:  $\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial p} \frac{\partial p}{\partial w_1} = \frac{\partial L}{\partial p} \frac{\partial p}{\partial h_2} \frac{\partial h_2}{\partial w_1}$

Для упрощения выкладок не будем копать глубоко в нейроны!

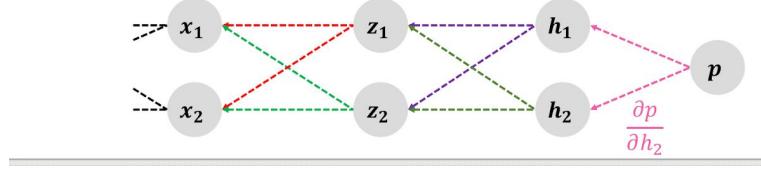


Это важное уточнение надо будет сделать если попадется билет. Так как дальше находят производные по значениям, а не по весам. Надо понимать, что любую функцию по значению можно продифференцировать по соответствующим весам и тогда получится нужный нам градиент.

Для большего понимания может помочь видео (рекомендую) [видео](#)

Вот это нужно для SGD

$$\begin{aligned} 3: \quad & \frac{\partial p}{\partial h_1} \quad \frac{\partial p}{\partial h_2} \\ 2: \quad & \frac{\partial p}{\partial z_1} = \frac{\partial p}{\partial h_1} \frac{\partial h_1}{\partial z_1} + \frac{\partial p}{\partial h_2} \frac{\partial h_2}{\partial z_1} \quad \frac{\partial p}{\partial z_2} = \frac{\partial p}{\partial h_1} \frac{\partial h_1}{\partial z_2} + \frac{\partial p}{\partial h_2} \frac{\partial h_2}{\partial z_2} \\ 1: \quad & \frac{\partial p}{\partial x_1} = \frac{\partial p}{\partial h_1} \frac{\partial h_1}{\partial z_1} \frac{\partial z_1}{\partial x_1} + \frac{\partial p}{\partial h_2} \frac{\partial h_2}{\partial z_1} \frac{\partial z_1}{\partial x_1} + \frac{\partial p}{\partial h_1} \frac{\partial h_1}{\partial z_2} \frac{\partial z_2}{\partial x_1} + \frac{\partial p}{\partial h_2} \frac{\partial h_2}{\partial z_2} \frac{\partial z_2}{\partial x_1} \\ & \frac{\partial p}{\partial x_2} = \frac{\partial p}{\partial h_1} \frac{\partial h_1}{\partial z_1} \frac{\partial z_1}{\partial x_2} + \frac{\partial p}{\partial h_2} \frac{\partial h_2}{\partial z_1} \frac{\partial z_1}{\partial x_2} + \frac{\partial p}{\partial h_1} \frac{\partial h_1}{\partial z_2} \frac{\partial z_2}{\partial x_2} + \frac{\partial p}{\partial h_2} \frac{\partial h_2}{\partial z_2} \frac{\partial z_2}{\partial x_2} \end{aligned}$$



10. Прямой и обратный проход для MLP с линейной активацией (матричные формулы) для одного примера

### Полносвязный слой как произведение матриц

- Пример для 2 нейронов с линейной активацией, 3 входами, без свободных членов:

$$(x_1 \quad x_2 \quad x_3) \cdot \begin{bmatrix} w_{1,1} & w_{1,2} \\ w_{2,1} & w_{2,2} \\ w_{3,1} & w_{3,2} \end{bmatrix} = (z_1 \quad z_2) \quad \begin{aligned} z_1 &= x_1 w_{1,1} + x_2 w_{2,1} + x_3 w_{3,1} \\ z_2 &= x_1 w_{1,2} + x_2 w_{2,2} + x_3 w_{3,2} \end{aligned}$$

$$xW = z$$

- Прямой проход:

$$xW = z \quad (x_1 \quad x_2 \quad x_3) \cdot \begin{bmatrix} w_{1,1} & w_{1,2} \\ w_{2,1} & w_{2,2} \\ w_{3,1} & w_{3,2} \end{bmatrix} = (z_1 \quad z_2)$$

- Для обратного прохода нужен  $\frac{\partial L}{\partial W}$ , где  $L(z_1, z_2)$  – скалярные потери.

$$\frac{\partial L}{\partial W} = \begin{bmatrix} \frac{\partial L}{\partial w_{1,1}} & \frac{\partial L}{\partial w_{1,2}} \\ \frac{\partial L}{\partial w_{2,1}} & \frac{\partial L}{\partial w_{2,2}} \\ \frac{\partial L}{\partial w_{3,1}} & \frac{\partial L}{\partial w_{3,2}} \end{bmatrix} \quad \text{Удобно для SGD:} \quad W_{new} = W - \gamma \frac{\partial L}{\partial W}$$

Перепишем в матричном виде:

$$\frac{\partial L}{\partial w_{i,j}} = \sum_k \frac{\partial L}{\partial z_k} \frac{\partial z_k}{\partial w_{i,j}} = \frac{\partial L}{\partial z_j} \frac{\partial z_j}{\partial w_{i,j}} = \frac{\partial L}{\partial z_j} x_i$$

$$\frac{\partial L}{\partial z} = \left( \frac{\partial L}{\partial z_1} \quad \frac{\partial L}{\partial z_2} \right) \quad \text{градиент}$$

$$\frac{\partial L}{\partial W} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \cdot \left( \frac{\partial L}{\partial z_1} \quad \frac{\partial L}{\partial z_2} \right) = x^T \frac{\partial L}{\partial z}$$

11. Прямой и обратный проход для MLP с линейной активацией (матричные формулы) для мини-батча (без производной по  $X$ )

## Прямой проход для мини-батча

$$\text{Батч из 2: } \begin{pmatrix} x_{1,1} & x_{1,2} & x_{1,3} \\ x_{2,1} & x_{2,2} & x_{2,3} \end{pmatrix} \cdot \begin{bmatrix} w_{1,1} & w_{1,2} \\ w_{2,1} & w_{2,2} \\ w_{3,1} & w_{3,2} \end{bmatrix} = \begin{pmatrix} z_{1,1} & z_{1,2} \\ z_{2,1} & z_{2,2} \end{pmatrix}$$

Матричный вид:  $XW = Z$

1 нейрон для 2 примера:  $z_{2,1} = x_{2,1}w_{1,1} + x_{2,2}w_{2,1} + x_{2,3}w_{3,1}$

## Обратный проход для мини-батча

$$\text{Батч из 2: } \begin{pmatrix} x_{1,1} & x_{1,2} & x_{1,3} \\ x_{2,1} & x_{2,2} & x_{2,3} \end{pmatrix} \cdot \begin{bmatrix} w_{1,1} & w_{1,2} \\ w_{2,1} & w_{2,2} \\ w_{3,1} & w_{3,2} \end{bmatrix} = \begin{pmatrix} z_{1,1} & z_{1,2} \\ z_{2,1} & z_{2,2} \end{pmatrix}$$

$$\text{SGD шаг: } \frac{\partial L_b}{\partial W} = \frac{\partial L(z_{1,1}, z_{1,2})}{\partial W} + \frac{\partial L(z_{2,1}, z_{2,2})}{\partial W}$$

$$\text{Для одного семпла: } \frac{\partial L}{\partial w_{i,j}} = \frac{\partial L}{\partial z_j} \frac{\partial z_j}{\partial w_{i,j}} = \frac{\partial L}{\partial z_j} x_i \quad \text{это уже знаем}$$

$$\text{Для 2 семплов: } \frac{\partial L_b}{\partial w_{i,j}} = \frac{\partial L}{\partial z_{1,j}} x_{1,i} + \frac{\partial L}{\partial z_{2,j}} x_{2,i}$$

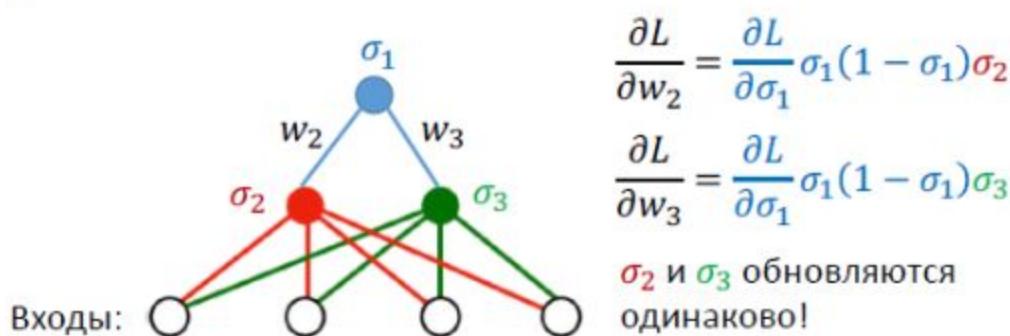
$$\frac{\partial L_b}{\partial W} = X^T \frac{\partial L}{\partial Z} \quad \left| \begin{array}{l} X^T = \begin{pmatrix} x_{1,1} & x_{2,1} \\ x_{1,2} & x_{2,2} \\ x_{1,3} & x_{2,3} \end{pmatrix} \\ \frac{\partial L}{\partial Z} = \begin{pmatrix} \frac{\partial L}{\partial z_{1,1}} & \frac{\partial L}{\partial z_{1,2}} \\ \frac{\partial L}{\partial z_{2,1}} & \frac{\partial L}{\partial z_{2,2}} \end{pmatrix} \end{array} \right.$$

$$\text{Для 2 семплов: } \frac{\partial L_b}{\partial w_{3,2}} = \frac{\partial L}{\partial z_{1,2}} x_{1,3} + \frac{\partial L}{\partial z_{2,2}} x_{2,3} \quad \text{Проверка!}$$

$$\frac{\partial L_b}{\partial W} = X^T \frac{\partial L}{\partial Z} \quad \left| \begin{array}{l} X^T = \begin{pmatrix} x_{1,1} & x_{2,1} \\ x_{1,2} & x_{2,2} \\ x_{1,3} & x_{2,3} \end{pmatrix} \\ \frac{\partial L}{\partial Z} = \begin{pmatrix} \frac{\partial L}{\partial z_{1,1}} & \frac{\partial L}{\partial z_{1,2}} \\ \frac{\partial L}{\partial z_{2,1}} & \frac{\partial L}{\partial z_{2,2}} \end{pmatrix} \end{array} \right.$$

Можно ли нулями и почему: НЕЛЬЗЯ  
Из каких соображений лучше выбирать масштаб случайных весов.

Давайте начнем с нулей?



Нужно сломать симметрию!

Может случайным шумом?

Но насколько большим?  $0.03 \cdot \mathcal{N}(0,1)$ ?

- Нейрон до активации:  $\sum_{i=1}^n x_i w_i$ .

- Если  $E(x_i) = E(w_i) = 0$  и мы генерируем веса независимо от входов, тогда  $E(\sum_{i=1}^n x_i w_i) = 0$ .

- Дисперсия до активации  $\sum_{i=1}^n x_i w_i$ :

$$Var(\sum_{i=1}^n x_i w_i) = \text{некоррелированные } x_i w_i \text{ и } x_j w_j \text{ из-за i.i.d. } w_k$$

$$= \sum_{i=1}^n Var(x_i w_i) = \text{ } w_i \text{ и } x_i \text{ независимы}$$

$$= \sum_{i=1}^n \left( [E(x_i)]^2 Var(w_i) + [E(w_i)]^2 Var(x_i) + Var(x_i) Var(w_i) \right) = \text{ } w_i \text{ и } x_i \text{ имеют 0 среднее}$$

$$= \sum_{i=1}^n Var(x_i) Var(w_i) = Var(x) [n Var(w)]$$

↑  
Хотим 1

- Для того, чтобы  $[n \operatorname{Var}(aw)]$  была 1  
нужно умножить  $\mathcal{N}(0,1)$  веса ( $\operatorname{Var}(w) = 1$ )  
на  $a = 1/\sqrt{n}$ .

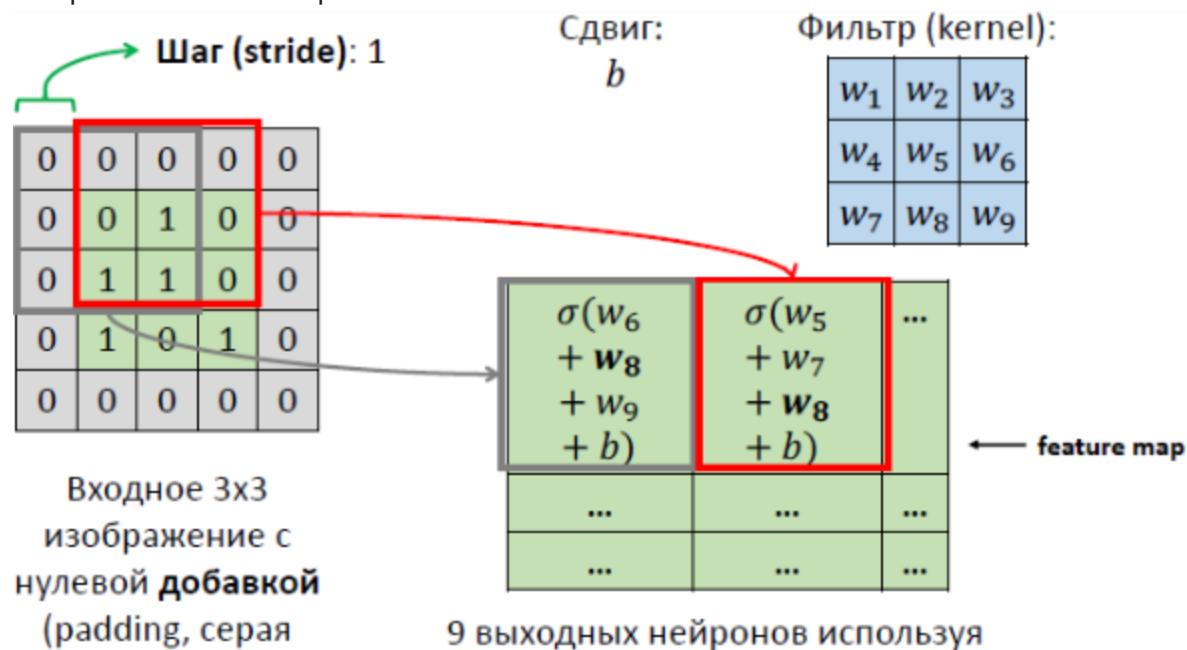
- Также известна как Xavier инициализация (Glorot et al.),  
в статье умножают на  $\sqrt{2}/\sqrt{n_{in} + n_{out}}$ .

- Инициализация для ReLU (He et al.) умножает на  $\sqrt{2}/\sqrt{n_{in}}$ .

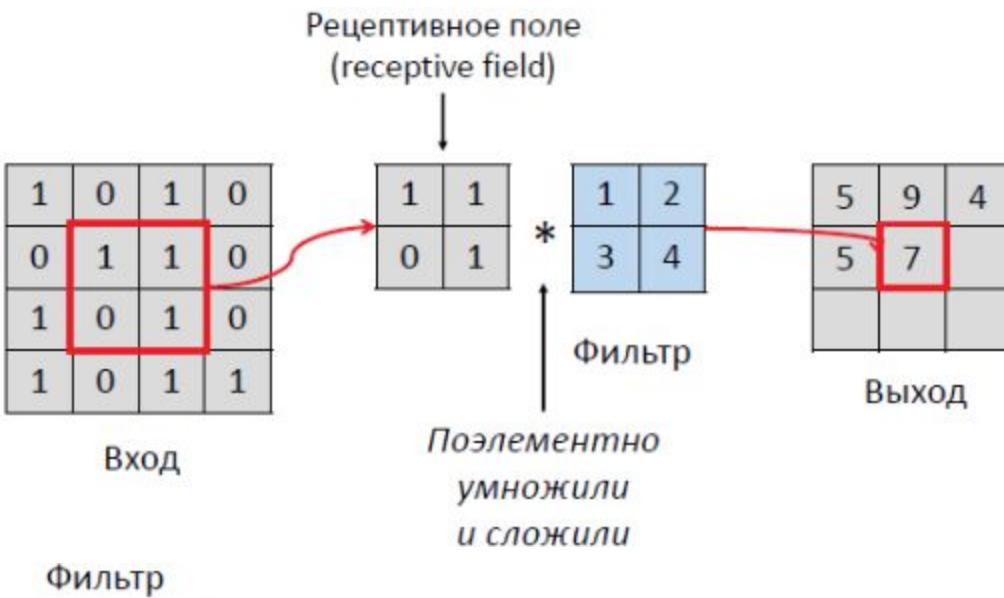
### 13. Сверточный слой для изображений.

Слой свёртки (*convolutional layer*) — это основной блок свёрточной нейронной сети. Слой свёртки включает в себя для каждого канала свой фильтр, ядро свёртки которого обрабатывает предыдущий слой по фрагментам (суммируя результаты матричного произведения для каждого фрагмента). Весовые коэффициенты ядра свёртки (небольшой матрицы) неизвестны и устанавливаются в процессе обучения. Особенностью свёрточного слоя является сравнительно небольшое количество параметров, устанавливаемое при обучении. Так например, если исходное изображение имеет размерность  $100 \times 100$  пикселей по трём каналам (это значит 30000 входных нейронов), а свёрточный слой использует фильтры с ядром  $3 \times 3$  пикселя с выходом на 6 каналов, тогда в процессе обучения определяется только 9 весов ядра, однако по всем сочетаниям каналов, то есть  $9 \times 3 \times 6 = 162$ , в таком случае данный слой требует нахождения только 162 параметров, что существенно меньше количества искомых параметров полно связной нейронной сети.

В нейросети, перед сверткой, вокруг изображения добавляют рамку из нулей. За счет этого размерность изображения после свертки не изменяется.



Примеры сверток.



-1	-1	-1
-1	8	-1
-1	-1	-1

- поиск краев.

0	-1	0
-1	5	-1
0	-1	0

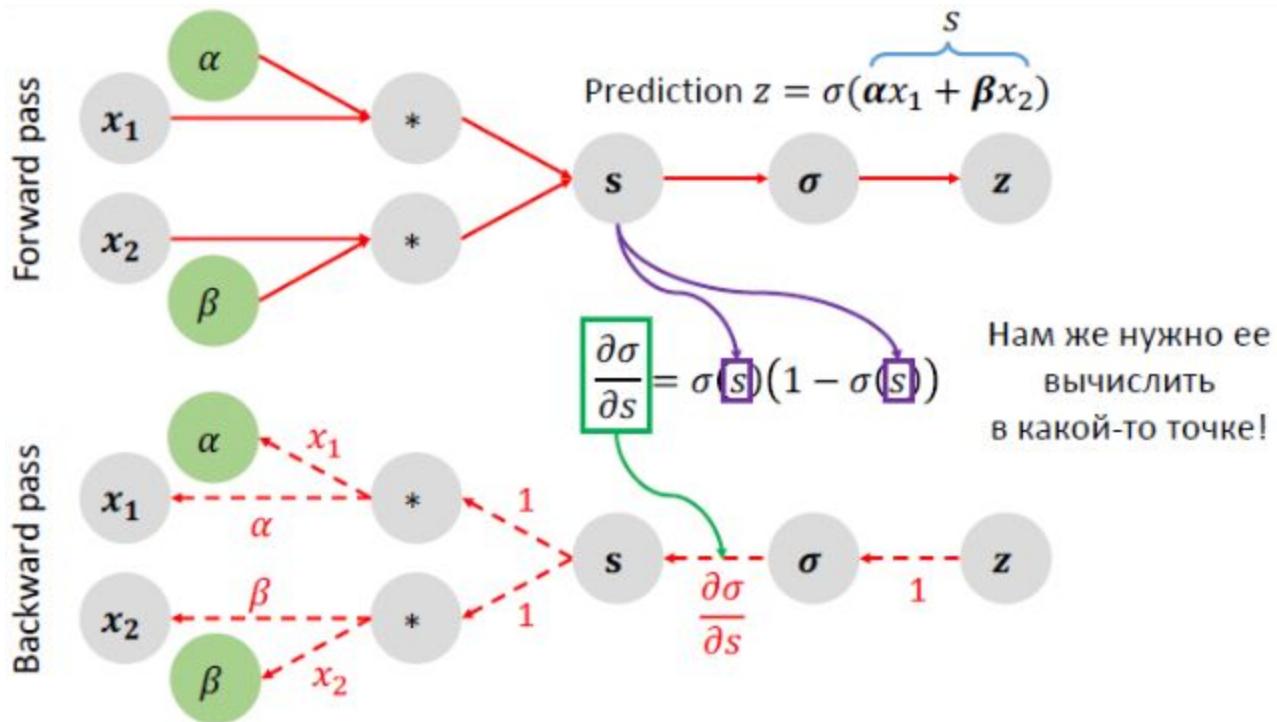
- повышение резкости.

$\frac{1}{9}$	1	1	1
1	1	1	1
1	1	1	1

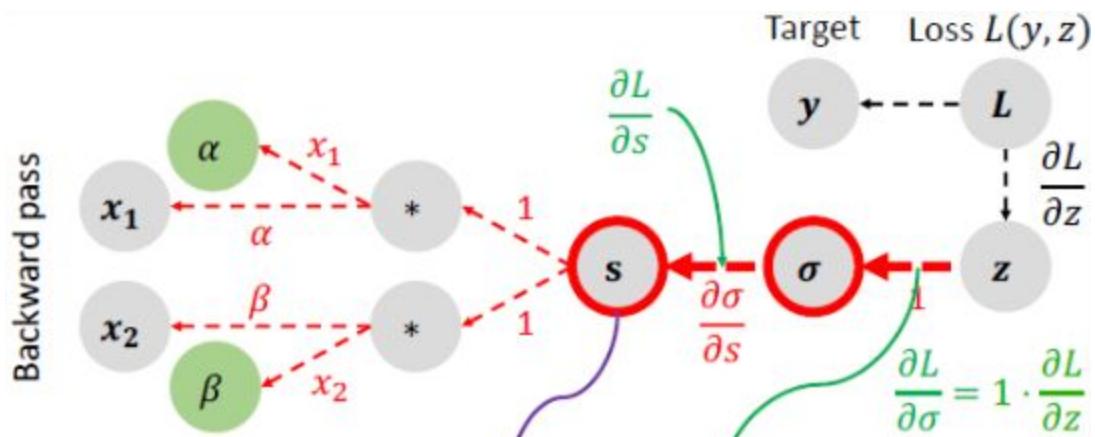
- размытие.

Как работает back-propagation для сверточного слоя.

Общий алгоритм для back-propagation:



```
def forward_pass(inputs):
    return 1. / (1 + np.exp(-inputs))
```

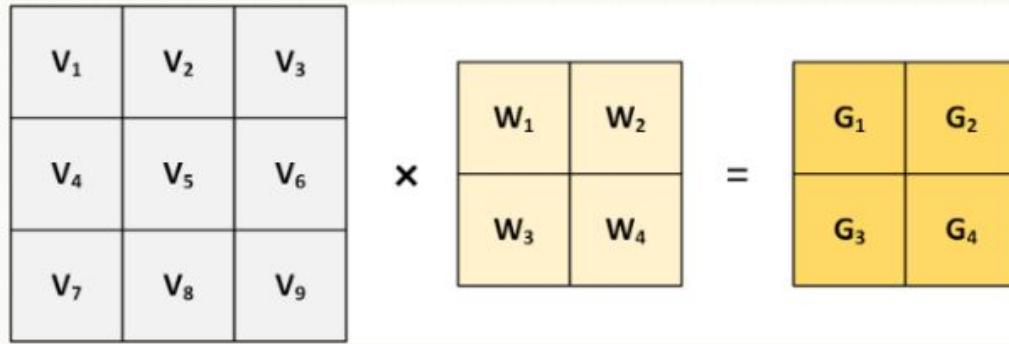


```
def backward_pass(inputs, incoming_gradient):
    sigmoid = 1. / (1 + np.exp(-inputs))
    return sigmoid * (1 - sigmoid) * incoming_gradient
```

$$\frac{\partial L}{\partial s} = \frac{\partial \sigma}{\partial s} \cdot \frac{\partial L}{\partial z}$$

Для слоя свертки:

The convolutional layer as described online.

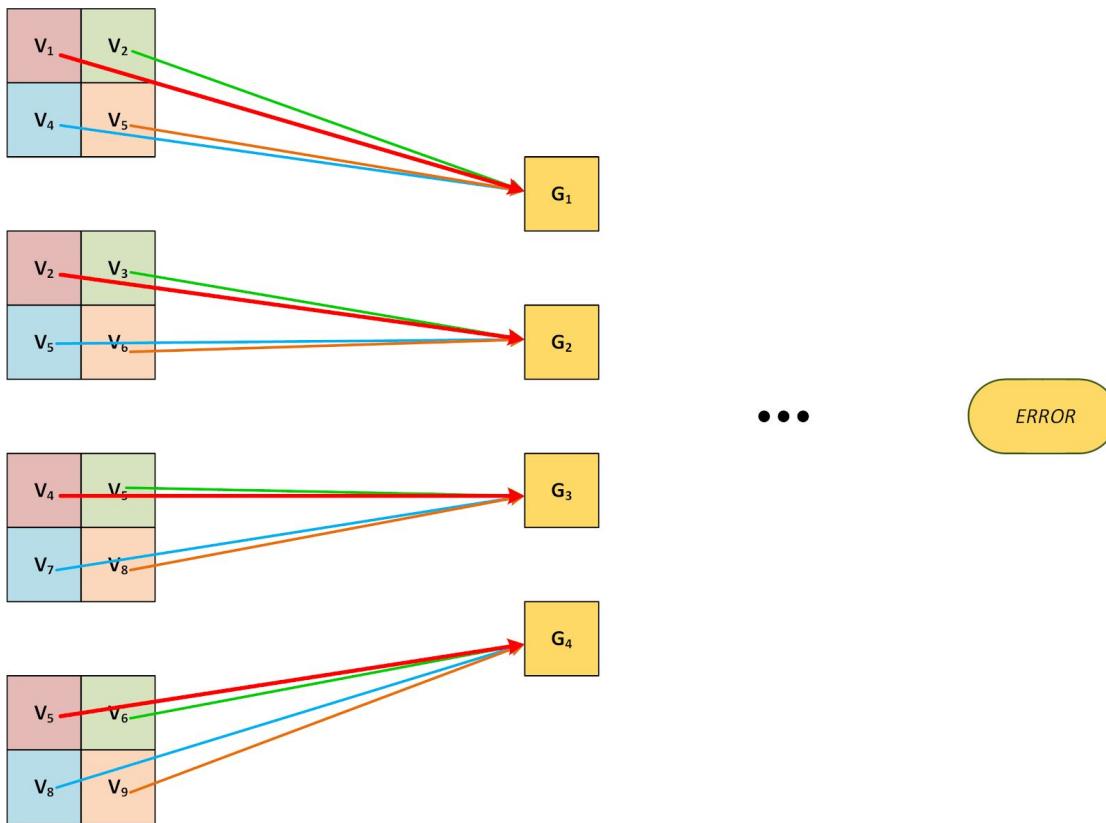


$$G_1 = V_1 W_1 + V_2 W_2 + V_4 W_3 + V_5 W_4$$

$$G_2 = V_2 W_1 + V_3 W_2 + V_5 W_3 + V_6 W_4$$

$$G_3 = V_4 W_1 + V_5 W_2 + V_7 W_3 + V_8 W_4$$

$$G_4 = V_5 W_1 + V_6 W_2 + V_8 W_3 + V_9 W_4$$



So, applying the chain rule as in the first example, we get the following:

$$\frac{\partial \text{Error}}{\partial W_1} = \frac{\partial \text{Error}}{\partial G_1} \frac{\partial G_1}{\partial W_1} + \frac{\partial \text{Error}}{\partial G_2} \frac{\partial G_2}{\partial W_1} + \frac{\partial \text{Error}}{\partial G_3} \frac{\partial G_3}{\partial W_1} + \frac{\partial \text{Error}}{\partial G_4} \frac{\partial G_4}{\partial W_1}$$

And the derivatives of interest ...

$$\frac{\partial G_1}{\partial W_1} = V_1$$

$$\frac{\partial G_2}{\partial W_1} = V_2$$

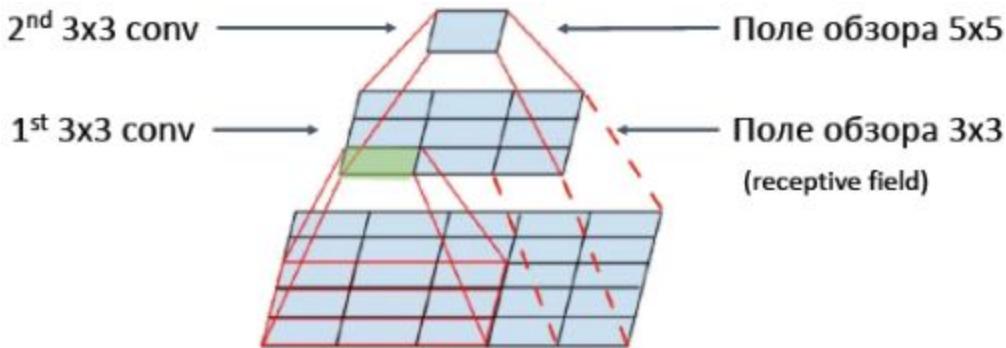
$$\frac{\partial G_3}{\partial W_1} = V_4$$

$$\frac{\partial G_4}{\partial W_1} = V_5$$

Error это тоже самое, что и Loss.

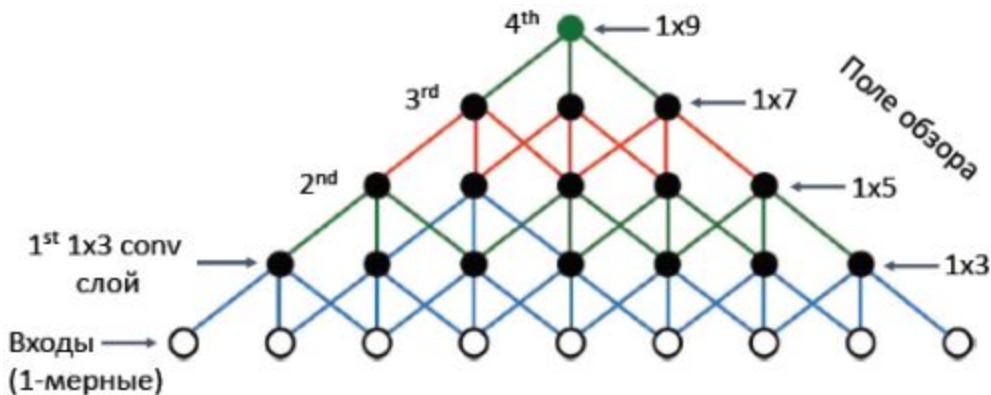
(<https://stats.stackexchange.com/questions/326377/backpropagation-on-a-convolutional-layer>)

#### 14. Что такое receptive field (поле обзора нейрона).

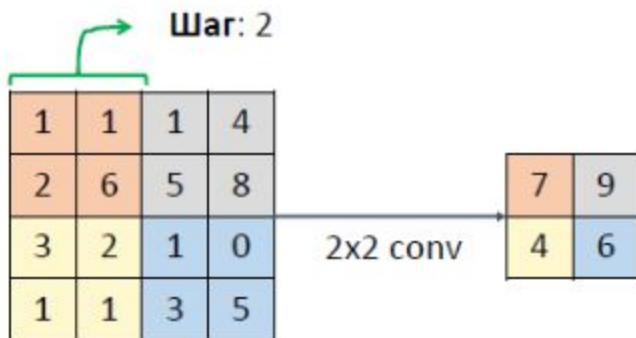


Объясните какими способами его можно увеличить в нейронной сети (два способа).

- 1) Добавить N сверточных слоев с фильтрами KxK



- 2) Увеличить шаг свертки



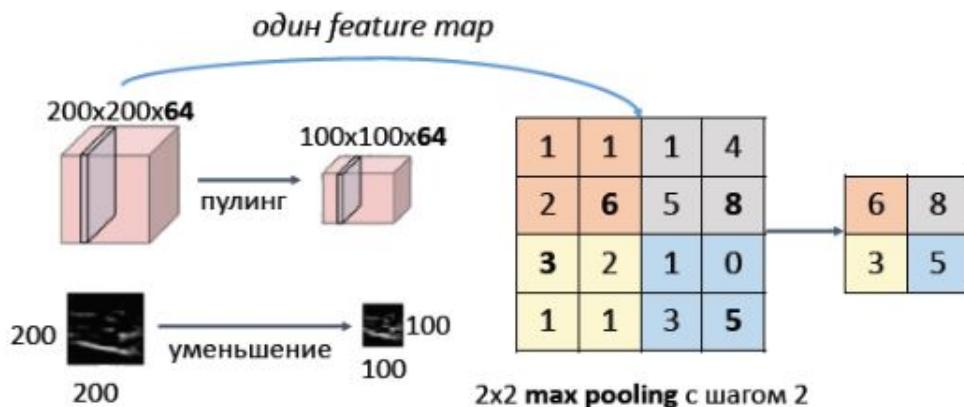
$$\begin{matrix} & \begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix} \\ * & \end{matrix}$$

Фильтр

- 3) Пулинг

## Пулинг слой

- Работает почти как свертка, только вместо свертки берет **максимум** (или среднее).



Как работает back-propagation для слоя max pooling.

## Max-Pooling Layer

Forward :

```
for (p = 0; p < k; p++)  
    for (q = 0; q < k; q++)  
         $y_n(x, y) = \max(y_n(x, y), y_{n-1}(x + p, y + q));$ 
```



Backward:

$$\frac{\partial L}{\partial y_{n-1}}(x + p, y + q) = \begin{cases} = 0, if (y_n(x, y) \neq y_{n-1}(x + p, y + q)) \\ = \frac{\partial L}{\partial y_n}(x, y), otherwise \end{cases}$$

([http://courses.cs.tau.ac.il/Caffe\\_workshop/Bootcamp/pdf\\_lectures/Lecture%203%20CNN%20-%20backpropagation.pdf](http://courses.cs.tau.ac.il/Caffe_workshop/Bootcamp/pdf_lectures/Lecture%203%20CNN%20-%20backpropagation.pdf))

Например, есть матрица

a	b
c	d

и maxpooling возвращает d. Функция maxpooling зависит только от d. Поэтому производная по d равна 1, а для a, b, c - ноль. Поэтому back-propagate 1 для чисел соответствующих d, и 0 для остальных.

(<https://stackoverflow.com/questions/40712031/backpropagation-in-pooling-layer-subsampling-layer-in-cnn>)

15. **Батч нормализация** пытается нормировать выходы нейронов (до и после активации).

- Нормализуем выход нейрона  $h_i$ :

$$h_i = \gamma_i \frac{h_i - \mu_i}{\sqrt{\sigma_i^2}} + \beta_i$$

среднее 0, дисперсия 1

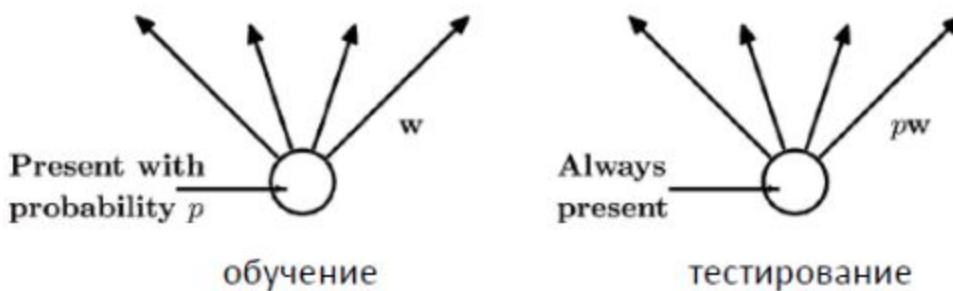
- Где взять  $\mu_i$  и  $\sigma_i^2$ ? Оценим их по **текущему батчу**!
- Во время тестирования будем использовать скользящее среднее:

$$0 < \alpha < 1 \quad \begin{aligned} \mu_i &= \alpha \cdot \text{mean}_{\text{batch}} + (1 - \alpha) \cdot \mu_i \\ \sigma_i^2 &= \alpha \cdot \text{variance}_{\text{batch}} + (1 - \alpha) \cdot \sigma_i^2 \end{aligned}$$

- Что делать с  $\gamma_i$  и  $\beta_i$ ? Линейная комбинация – это дифференцируемая операция, применим **backpropagation**!

**Дропаут:** оставляем нейрон с вероятностью  $p$ , а иначе выключаем (замена на 0). Таким образом мы сэмплируем сеть на каждом шаге обучения и меняем не все параметры => регуляризует. Во время тестирования все нейроны включены, но умножены на  $p$  для поддержания масштаба.

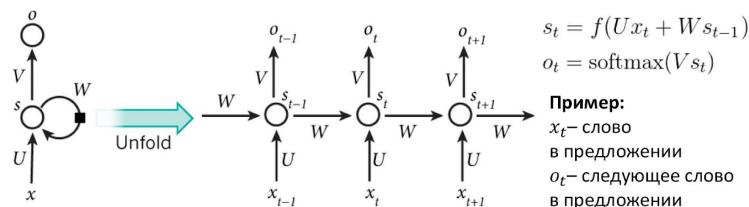
Ожидаемый вес:  $p \cdot w + (1 - p) \cdot 0$



16. Устройство простой рекуррентной сети (две формулы с объяснением мотивации). Производная скрытого состояния  $s_t$  по  $W$ . Применение для задачи language modeling.

Две нужные формулы в верхнем правом углу. Грубо говоря  $s_t \leftarrow$  это значение до функции активации. Оно

### RNN для LM



- Работает **одинаково** для каждого элемента последовательности  $x_t$ , но вычисления зависят от предыдущих элементов  $x_t$
- Можно сказать, что у RNN есть память (**скрытое состояние**  $s_t$ ), в которой хранится информация о предыдущих элементах последовательности

равно сумме  $X^*U \leftarrow$  т.е. то, что пришло на вход в текущую ячейку, а  $W * S(t-1)$  это слагаемое отвечает за предыдущее состояние. На это все мы вешаем soft-max функцию и получаем значение выхода.

Почему подходит для обработки языка? Потому что классическая задача ставится через условную

### Производная для $s_2$

Обозначим аргумент:

$$s_1 = f(Ux_1 + Ws_0)$$

$$s_2 = f(Ux_2 + Ws_1) = f(Ux_2 + Wf(Ux_1 + Ws_0))$$

$$\frac{\partial s_2}{\partial W} = \frac{\partial f}{\partial a_2} \left( W \frac{\partial s_1}{\partial W} + s_1 \right) = \boxed{\frac{\partial f}{\partial a_2}} W \frac{\partial s_1}{\partial W} + \boxed{\frac{\partial f}{\partial a_2} s_1}$$

Потому что  $s_0$  константа:

$$\frac{\partial s_1}{\partial W} = \frac{\partial s_1}{\partial W_*}$$

$$\frac{\partial s_2}{\partial s_1}$$

$$\frac{\partial s_2}{\partial W_*}$$

Результат:

$$\frac{\partial s_2}{\partial W} = \frac{\partial s_2}{\partial s_1} \frac{\partial s_1}{\partial W_*} + \frac{\partial s_2}{\partial W_*}$$

Обозначение для  $s_{j=2}$  в предположении, что  $s_{j-1}$  не зависит от  $W$

вероятность.

## Держим в голове пример: Language Model

Генеративная модель естественного языка:

$$P(\text{text}) = P(x_0, \dots, x_n) = \\ = P(x_0)P(x_1|x_0)P(x_2|x_0, x_1) \dots P(x_n| \dots )$$

А наша RNN сеть тоже самое и представляет, то есть каждый следующий нейрон зависит от предыдущего.

17. Производная скрытого состояния  $s_3$  по  $W$ . Общий вид производной для  $s_k$ .

Тут достаточно легко. Сравниваем производную для  $s_2$  и находим общий смысл этого всего.

По индукции...

$$\frac{\partial s_2}{\partial W} = \frac{\partial s_2}{\partial s_1} \frac{\partial s_1}{\partial W_*} + \frac{\partial s_2}{\partial W_*}$$

$$\frac{\partial s_3}{\partial W} = \frac{\partial s_3}{\partial s_2} \frac{\partial s_2}{\partial s_1} \frac{\partial s_1}{\partial W_*} + \frac{\partial s_3}{\partial s_2} \frac{\partial s_2}{\partial W_*} + \frac{\partial s_3}{\partial W_*}$$

$$\frac{\partial s_k}{\partial W} = \sum_{i=1}^k \left( \prod_{j=i+1}^k \frac{\partial s_j}{\partial s_{j-1}} \right) \frac{\partial s_i}{\partial W_*}$$

$$\frac{\partial s_k}{\partial W} = \frac{\partial s_k}{\partial s_{k-1}} \frac{\partial s_{k-1}}{\partial s_{k-2}} \dots \frac{\partial s_2}{\partial s_1} \frac{\partial s_1}{\partial W_*} + \frac{\partial s_k}{\partial s_{k-1}} \frac{\partial s_{k-1}}{\partial W_*} + \frac{\partial s_k}{\partial W_*}$$

$s_{j-1}$  не зависит от  $W$

Результат

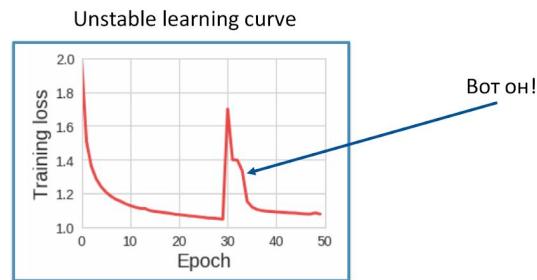
18. Проблема затухающих или взрывающихся градиентов (показать на примере производной для  $s_k$  по  $W$ ). Что делать в случае затухания градиентов и что делать в случае взрывающихся градиентов?

В чем проблема? Если градиент “взрывается”, то наш градиентный спуск шагнет не туда. И это можно будет легко понять, например, по графику функции потерь.

Как бороться? - Ограничить норму весов градиента.

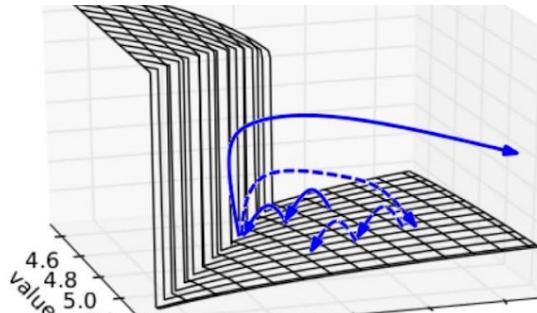
Проблема затухающих градиентов:

### Взрыв легко заметить на графике



Возникает тогда, когда много нейронов. То есть мы же считаем градиент через предыдущие нейроны. И когда доходит дело до изменения последних слоев, то они практически не изменяются. Это плохо так как последние слои чаще всего остаются недообученными, а быстрее всего обучаются слои, которые ближе к

```
 $\hat{g} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$ 
if  $\|\hat{g}\| \geq threshold$  then
     $\hat{g} \leftarrow \frac{threshold}{\|\hat{g}\|} \hat{g}$ 
end if
```

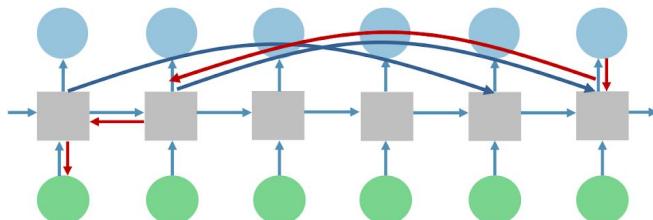


выходу.

Как бороться? - Пробрасывать градиенты через нейроны, чтобы градиент не успел затухнуть. Такой подход называется skip connections. Также можно нарисовать такую картинку

Другим способом борьбы с затухающими градиентами служит усовершенствование ячейка - LSTM. Не думаю, что ее надо расписывать, но сказать, что там есть проход из любой ячейки для градиента в другую. Именно это ( и не только ) позволяет бороться с проблемой затухающих градиентов.

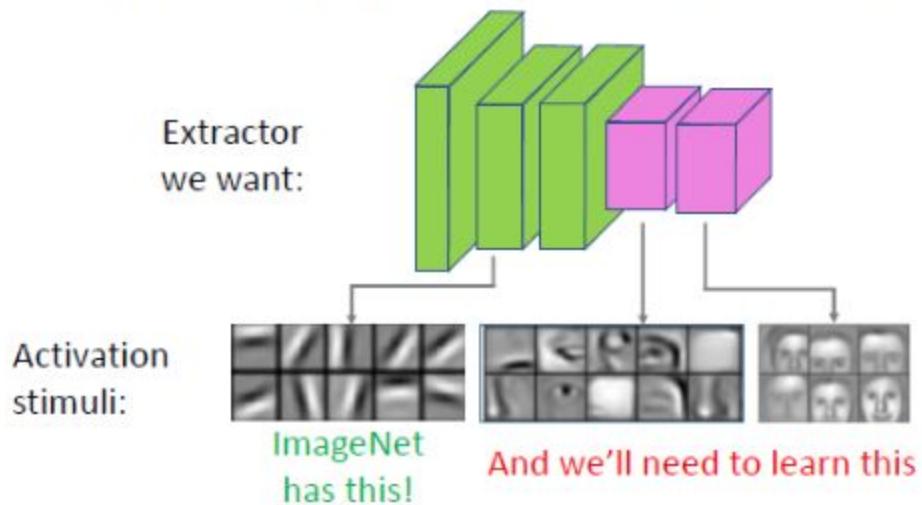
### Что делать с затуханием



Градиенты распространяются дальше!

19. Transfer learning и fine-tuning. Когда нужно использовать.

- But what if we need to classify human emotions?
- Maybe we can partially reuse ImageNet features extractor?

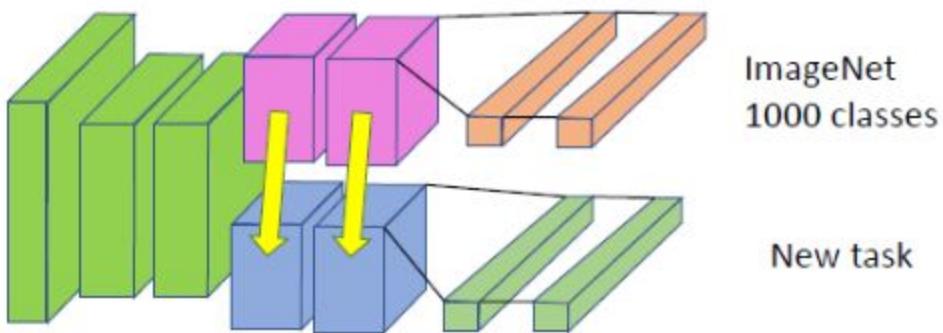


Чтобы обучить глубокие сети нужно много данных. Что, если мы можем повторно использовать существующие для новой задачи?

- 1) Нужно меньше данных для тренировки
- 2) Это работает если область новой задачи похожа на ImageNet
- 3) Это не будет работать для классификаций человеческих эмоций, ImageNet не имеет в памяти человеческих лиц.

Но что если нам нужно классифицировать человеческие эмоции? Возможно мы можем использовать ImageNet частично?

#### ImageNet features extractor



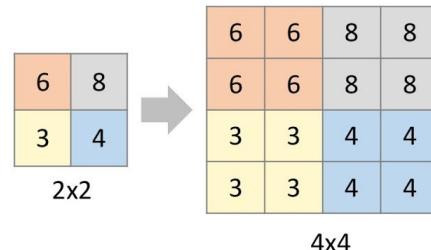
Мы можем инициализировать глубокие слои, используя данные из ImageNet. Это называется fine-tuning, т.к. мы начинаем не со случайных начальных условий.

- 1) Распространяются все градиенты с меньшей скоростью обучения.
- 2) Очень часто используется благодаря широкому спектру классов ImageNet
- 3) Keras имеет вес предварительно подготовленных архитектур VGG, Inception, ResNet
- 4) Вы можете точно настроить кучу разных архитектур и сделать из них ансамбль!

20. Варианты реализации un-pooling слоев (по соседям, максимальные индексы, transpose convolution) и мотивация

## Nearest neighbor unpooling

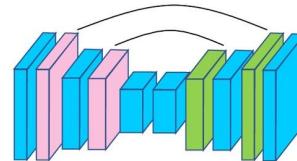
Fill with nearest neighbor values



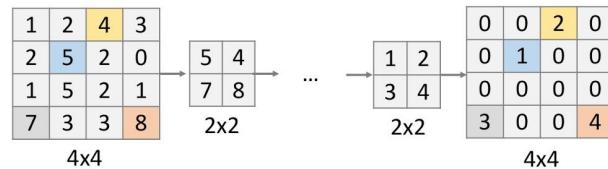
Pixelated and not crisp!

## Max unpooling

Corresponding pairs of  
downsampling and  
upsampling layers



Remember which element was max during pooling, and fill that position during unpooling:

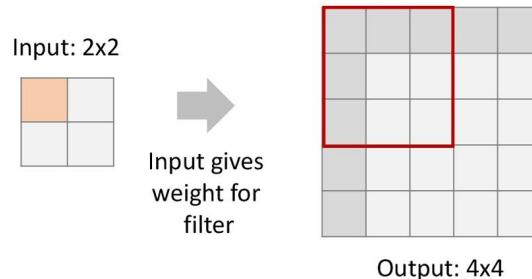


## Learnable unpooling

- Previous approaches are not data-driven!
- We can replace max pooling layer with convolutional layer that has a bigger stride!
- What if we can apply convolutions to do unpooling?

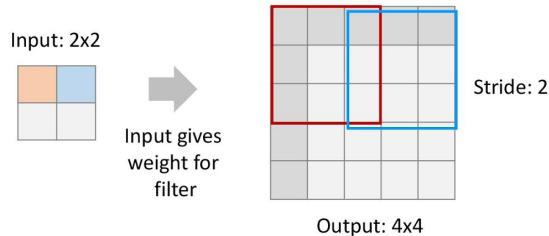
## Learnable unpooling

- Previous approaches are not data-driven!
- We can replace max pooling layer with convolutional layer that has a bigger stride!
- What if we can apply convolutions to do unpooling?



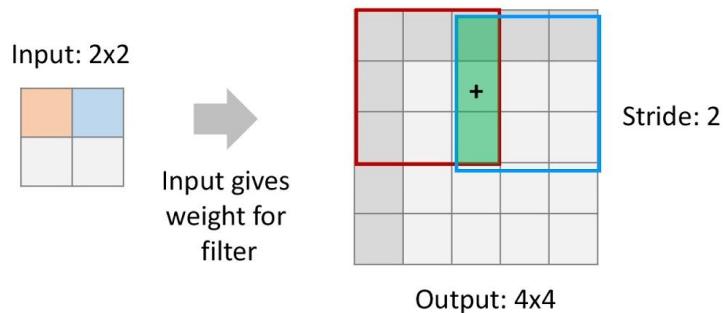
## Learnable unpooling

- Previous approaches are not data-driven!
- We can replace max pooling layer with convolutional layer that has a bigger stride!
- What if we can apply convolutions to do unpooling?

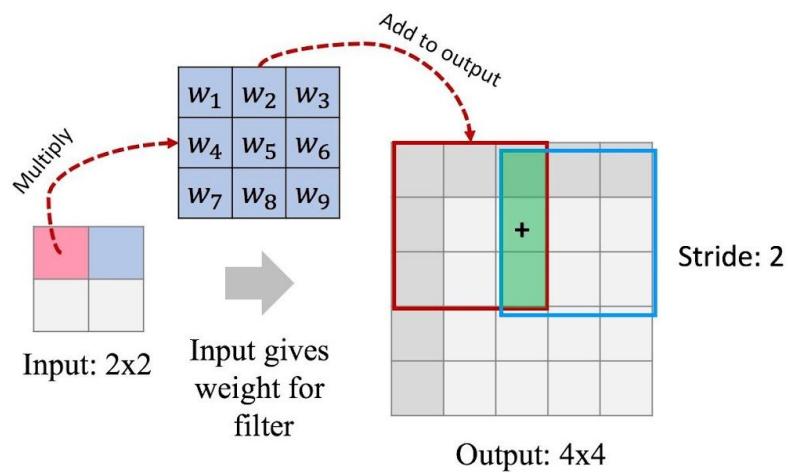


## Learnable unpooling

- Previous approaches are not data-driven!
- We can replace max pooling layer with convolutional layer that has a bigger stride!
- What if we can apply convolutions to do unpooling?

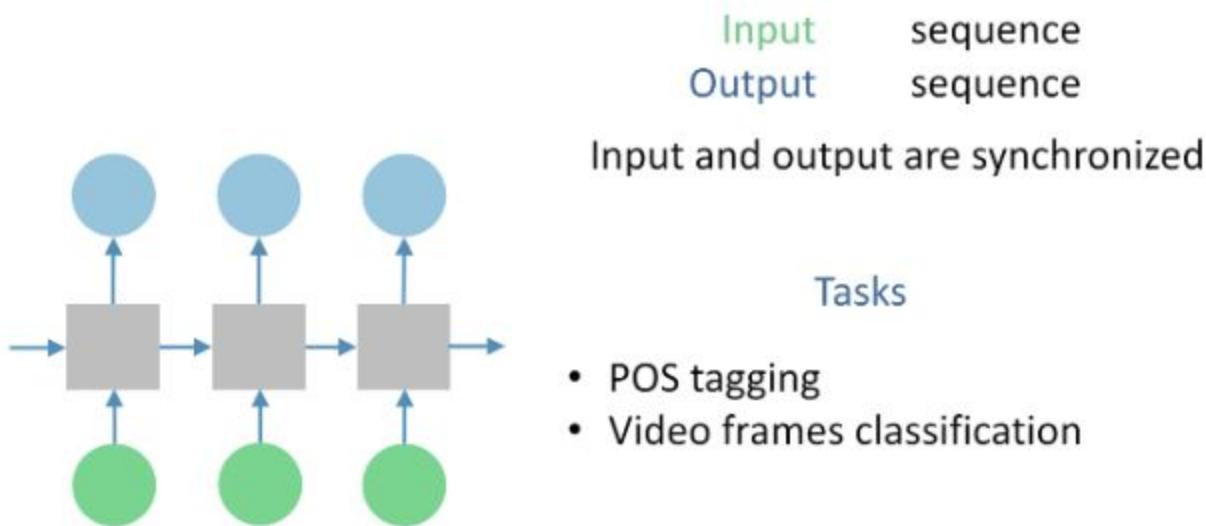


## Transpose convolution

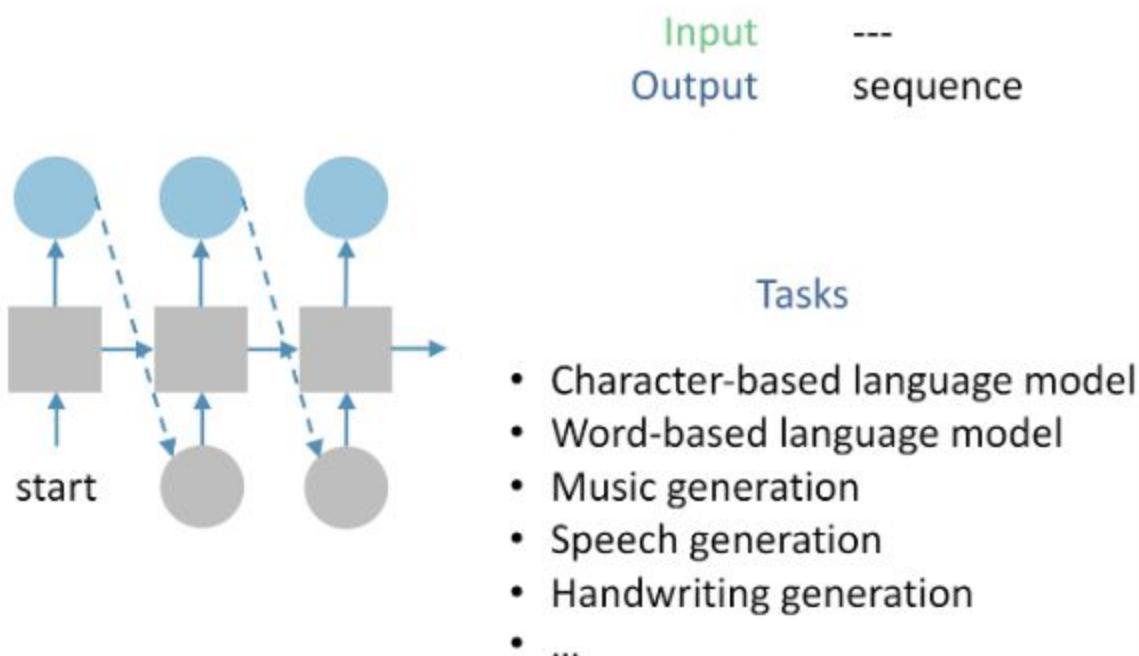


21. Примеры рекуррентных архитектур для разных задач

# Elements-wise classification

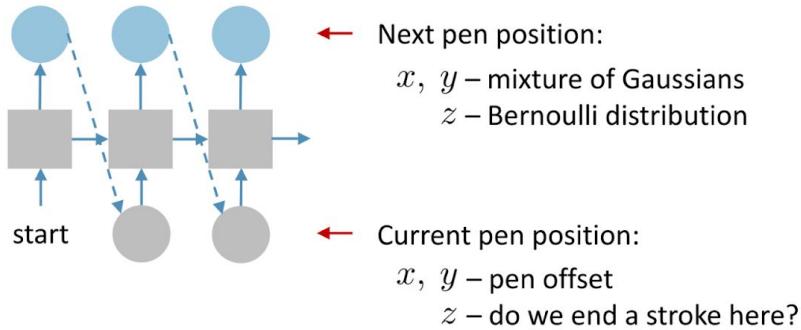


# Sequence generation

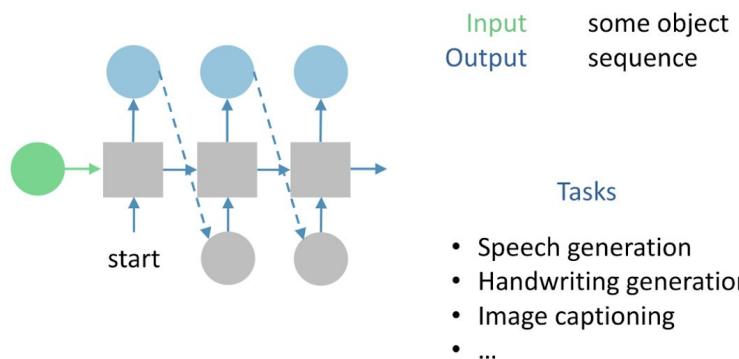


# Handwriting generation

We predict handwriting point by point

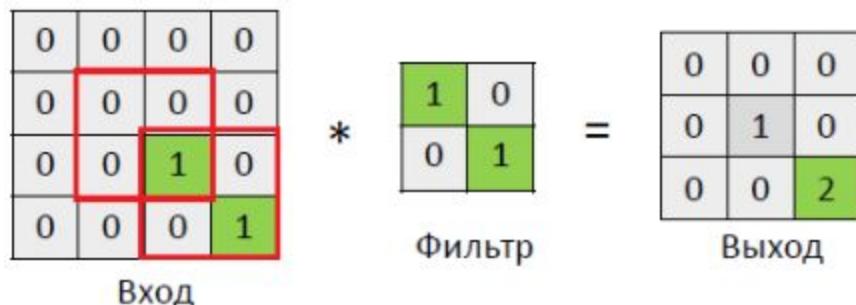


# Conditional sequence generation



Примеры задач:

1. Посчитать для простой (3x3) картинки операцию свертки.



Пример для картинки 4x4:



Возможно нужно сделать рамку вокруг изображения:

2. Взять производную сложной функции (например,  $d[(xy + e^{txy})^y] / dy$ )

<b>Простые</b>		<b>Сложные</b>	
$y = x^n$	$y' = nx^{n-1}$	$y = U^\alpha$	$y' = \alpha U^{\alpha-1}U'$
$y = \frac{1}{x}$	$y' = -\frac{1}{x^2}$	$y = \frac{1}{U}$	$y' = -\frac{U'}{U^2}$
$y = \sqrt{x}$	$y' = \frac{1}{2\sqrt{x}}$	$y = \sqrt{U}$	$y' = \frac{U'}{2\sqrt{U}}$
$y = \sin x$	$y' = \cos x$	$y = \sin U$	$y' = (\cos U) \cdot U'$
$y = \cos x$	$y' = -\sin x$	$y = \cos U$	$y' = (-\sin x) \cdot U'$
$y = \operatorname{tg} x$	$y' = -\frac{1}{\cos^2 x}$	$y = \operatorname{tg} U$	$y' = -\frac{U'}{\cos^2 U}$
$y = \operatorname{ctg} x$	$y' = -\frac{1}{\sin^2 x}$	$y = \operatorname{ctg} U$	$y' = -\frac{U'}{\sin^2 U}$
$y = a^x$	$y' = a^x \ln a$	$y = a^U$	$y' = U^x \ln U \cdot U'$
$y = e^x$	$y' = e^x$	$y = e^U$	$y' = e^U \cdot U'$
$y = \log_a x$	$y' = \frac{1}{x \ln a}$	$y = \log_a U$	$y' = \frac{U'}{U \ln a}$
$y = \ln x$	$y' = \frac{1}{x}$	$y = \ln U$	$y' = \frac{U'}{U}$
$y = \arccos x$	$y' = -\frac{1}{\sqrt{1-x^2}}$	$y = \arccos U$	$y' = -\frac{U'}{\sqrt{1-U^2}}$
$y = \arcsin x$	$y' = \frac{1}{\sqrt{1-x^2}}$	$y = \arcsin U$	$y' = \frac{U'}{\sqrt{1-U^2}}$
$y = \operatorname{arctg} x$	$y' = \frac{1}{1+x^2}$	$y = \operatorname{arctg} U$	$y' = \frac{U'}{1+U^2}$
$y = shx$	$y' = chx$	$y = shU$	$y' = chU \cdot U'$
$y = chx$	$y' = shx$	$y = chU$	$y' = shU \cdot U'$
$y = thx$	$y' = \frac{1}{ch^2 x}$	$y = thU$	$y' = \frac{U'}{ch^2 U}$
$y = cthx$	$y' = -\frac{1}{sh^2 x}$	$y = cthU$	$y' = -\frac{U'}{sh^2 U}$