



Data Mining in Action

Лекция 7. Нейронные сети: часть 1

Партнеры курса



Национальный исследовательский
технологический университет

misis.ru



jet.su

На прошлой лекции

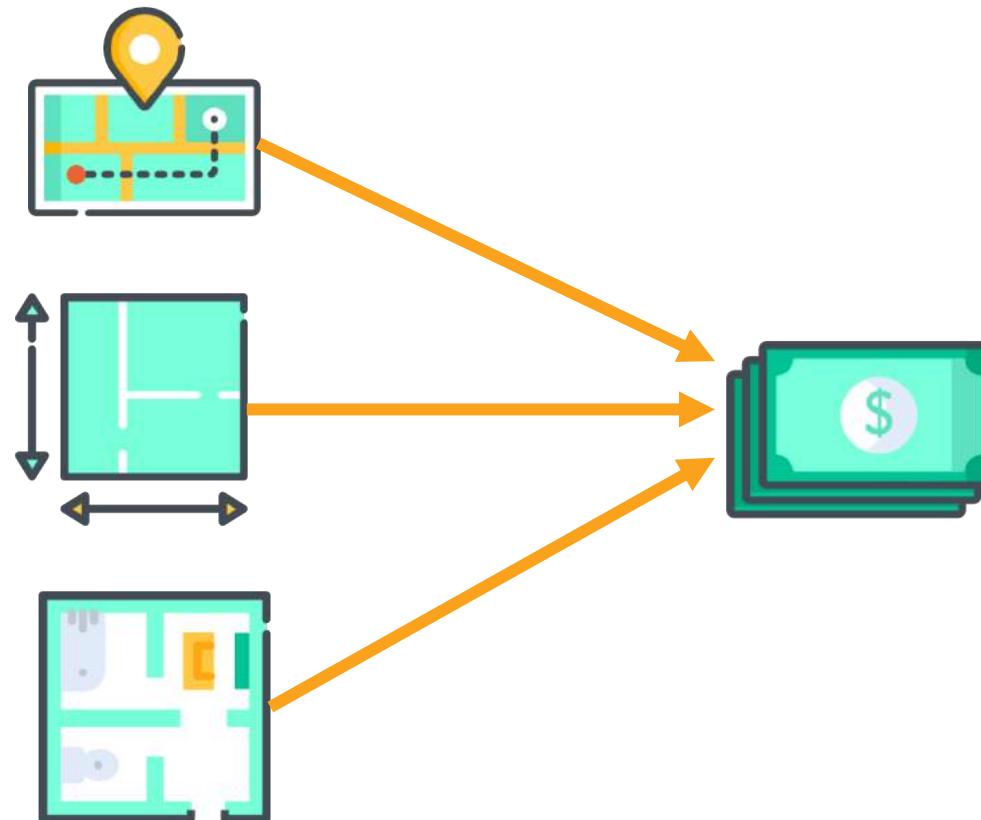
- Кластеризация
- Понижение размерности
- Матричные разложения
- Обучение векторных представлений

План

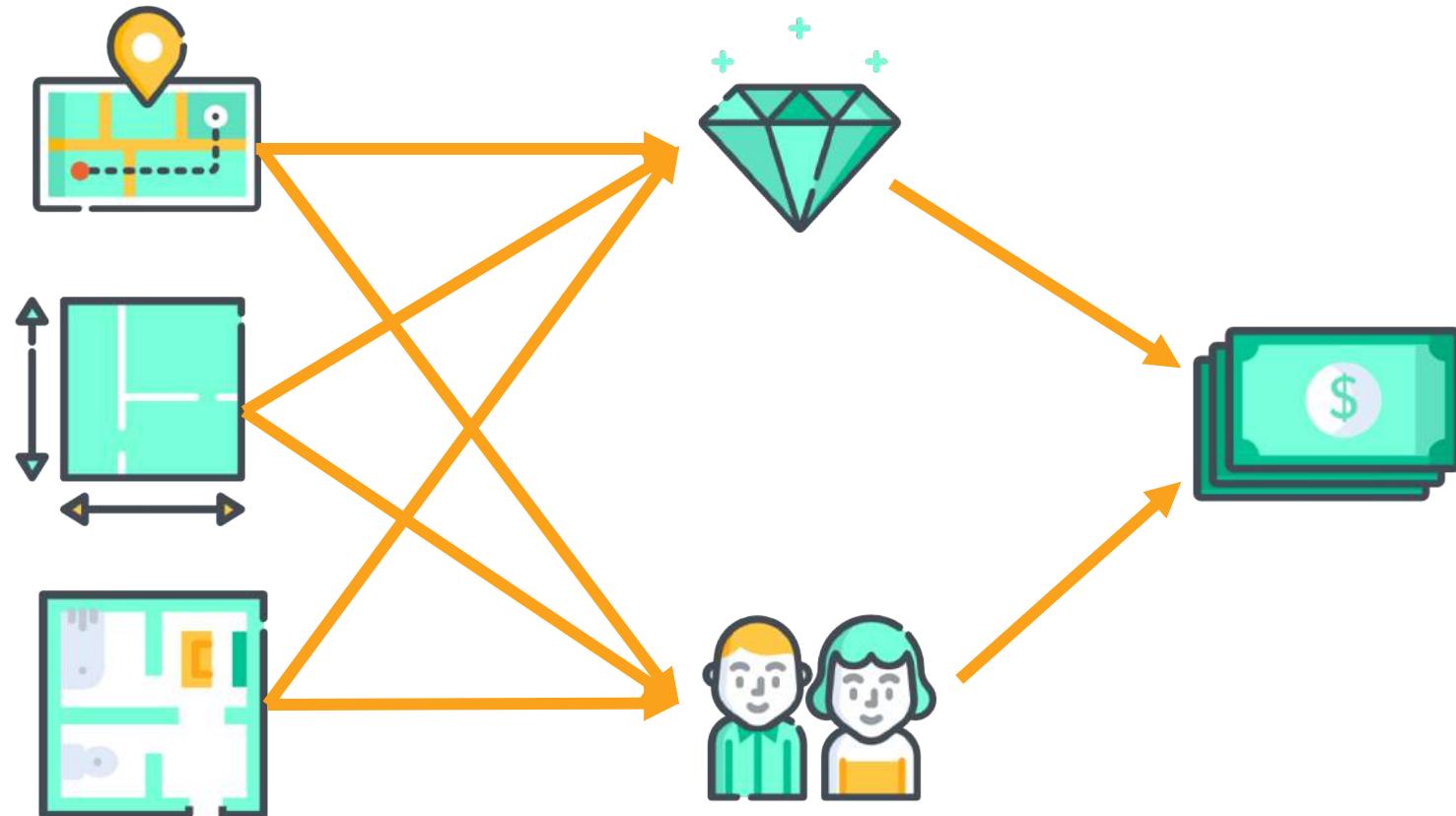
1. Что такое нейросети
2. Обучение нейросетей
3. Свёрточные нейросети

1. Что такое нейросети

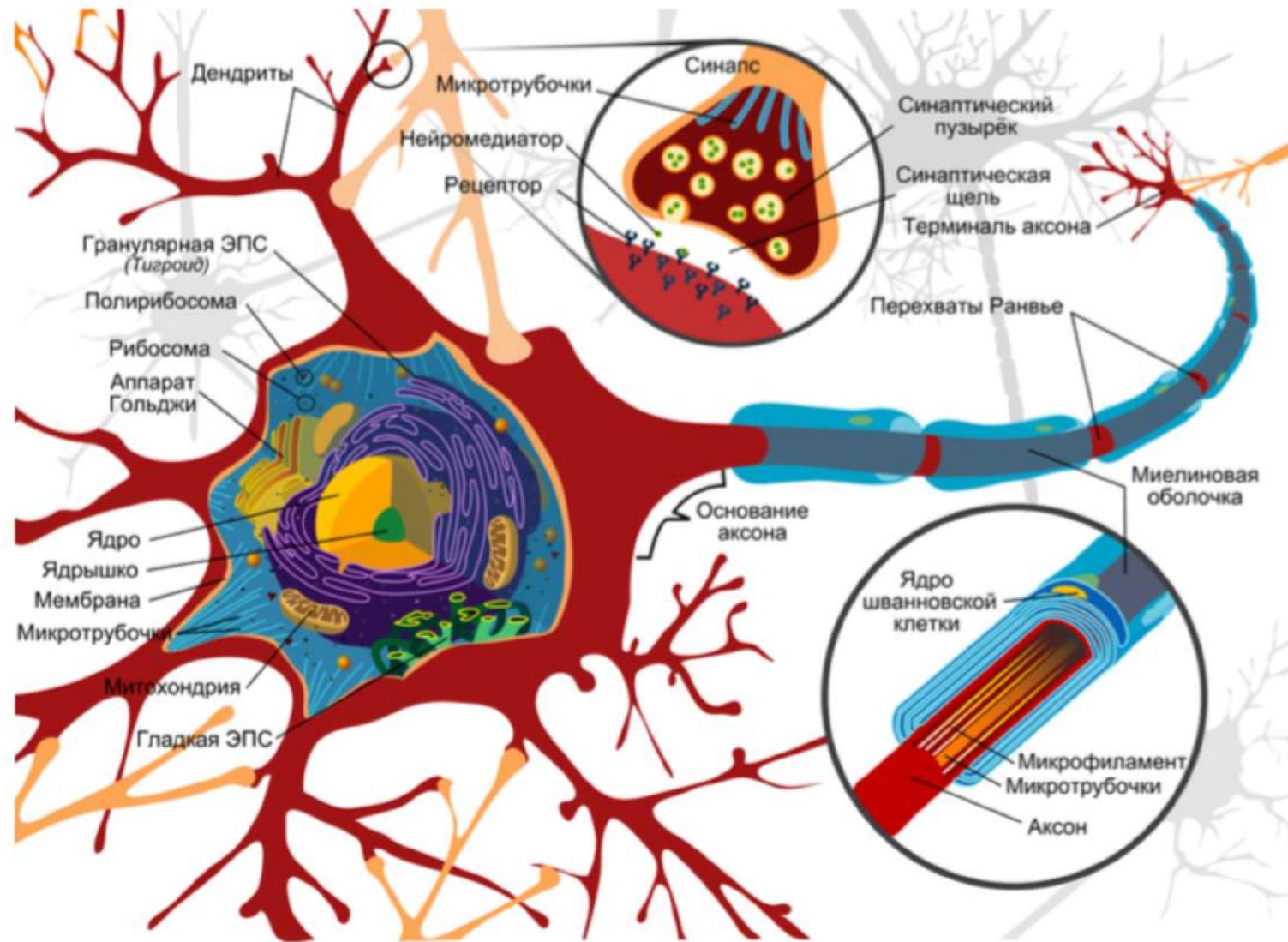
Пример: прогнозирование цен на недвижимость



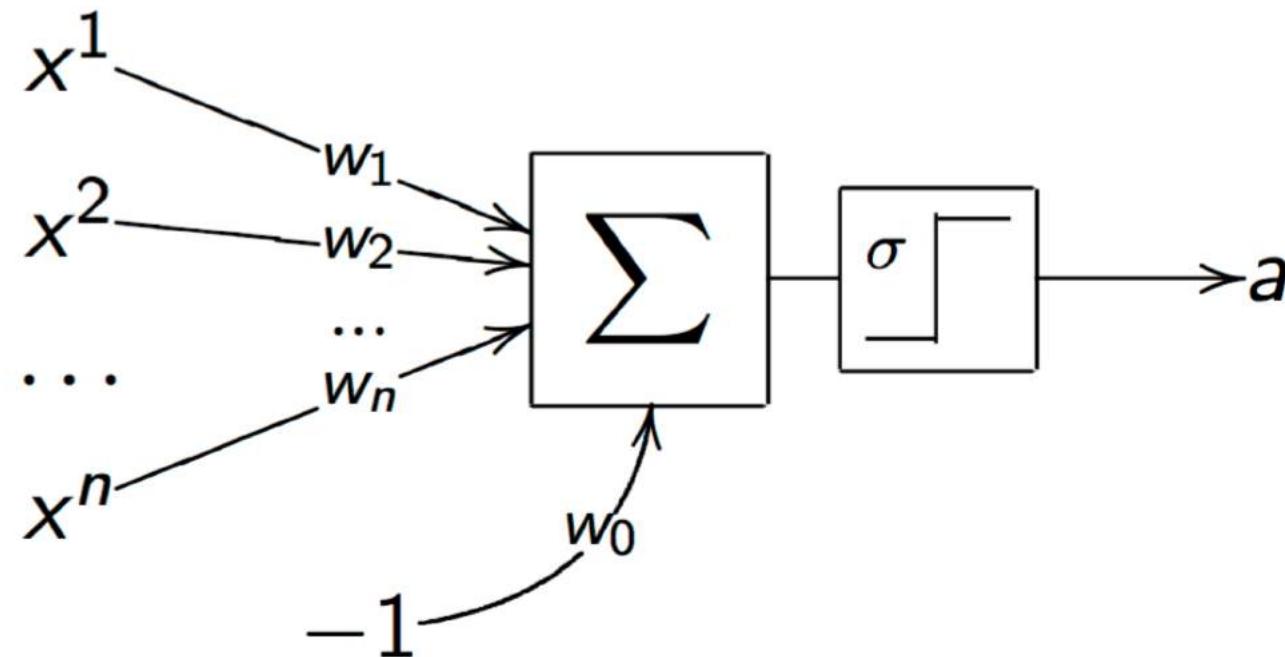
Пример: прогнозирование цен на недвижимость



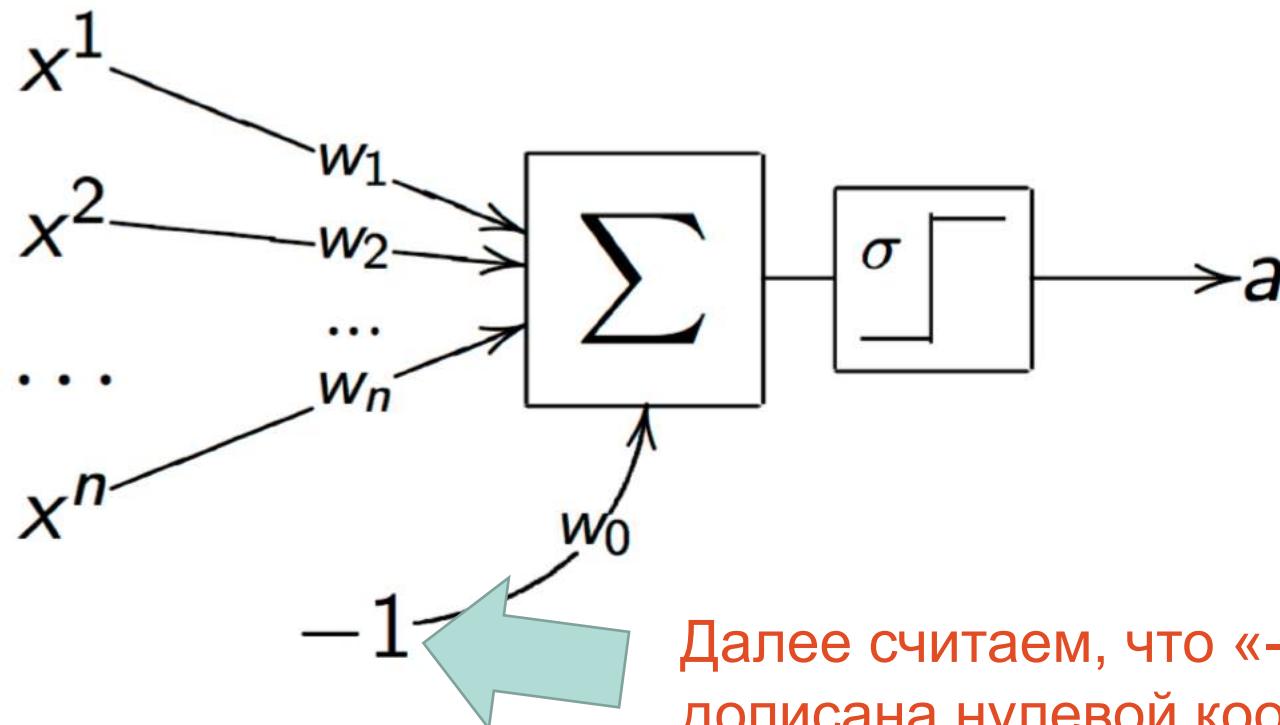
Нейрон в биологии



Нейрон в машинном обучении

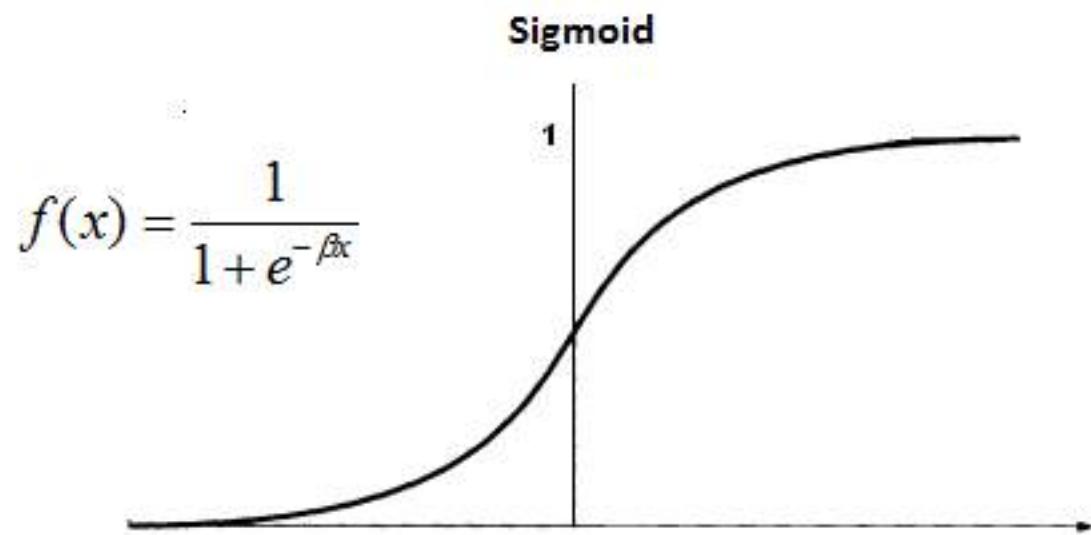


Модель нейрона

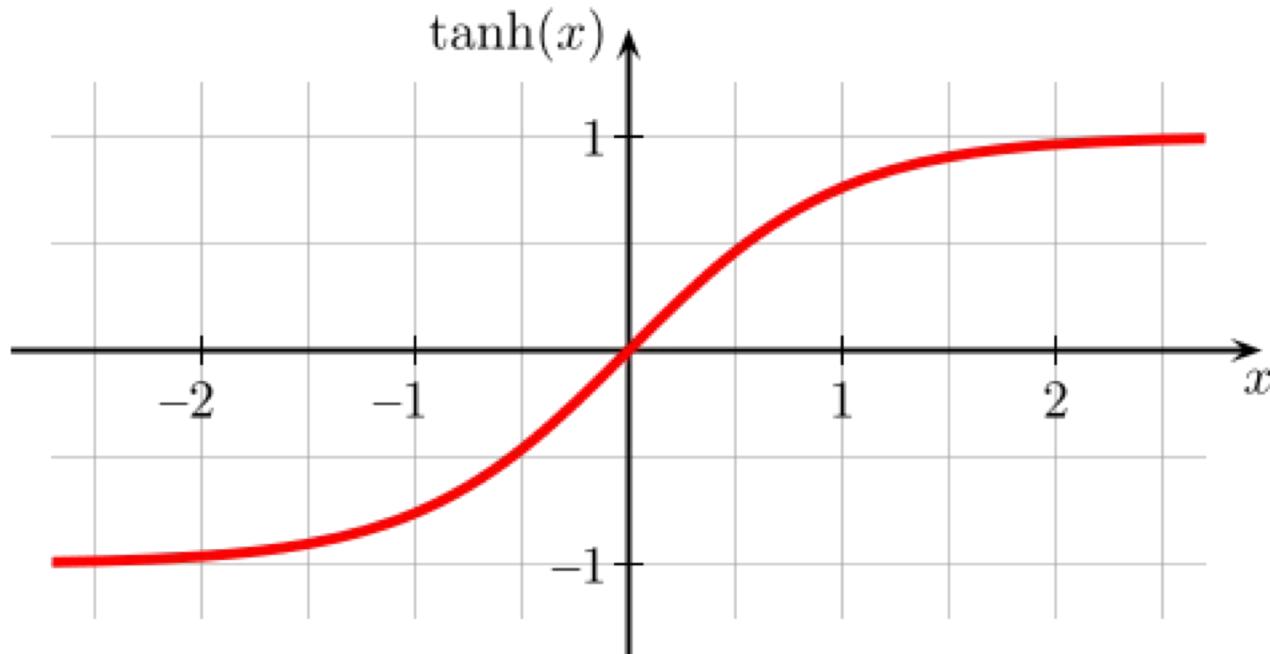


Далее считаем, что «-1» всегда
дописана нулевой координатой в
векторе признаков и постоянно
выписывать порог W_0 уже не будем

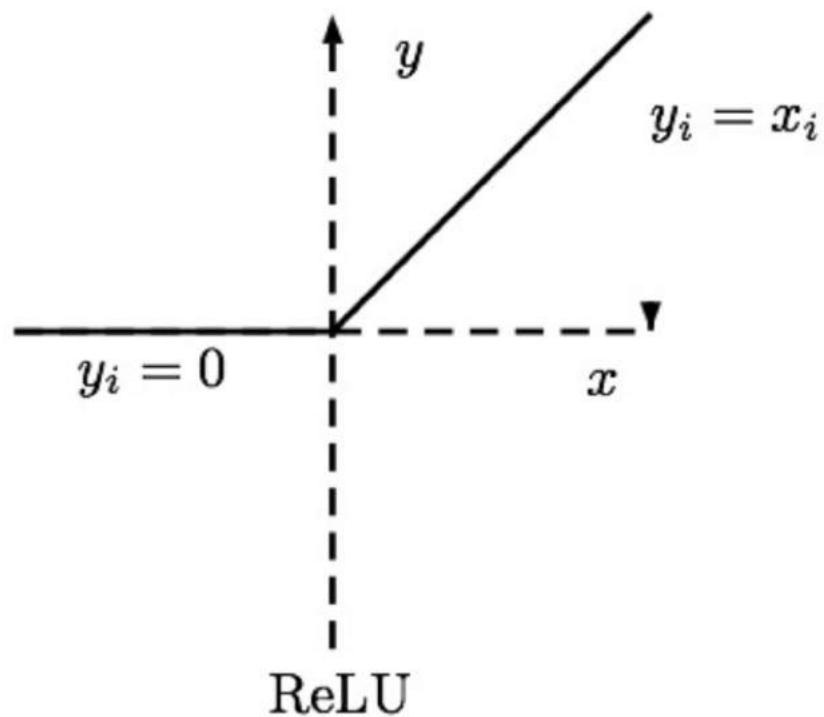
Функции активации (нелинейности): sigmoid



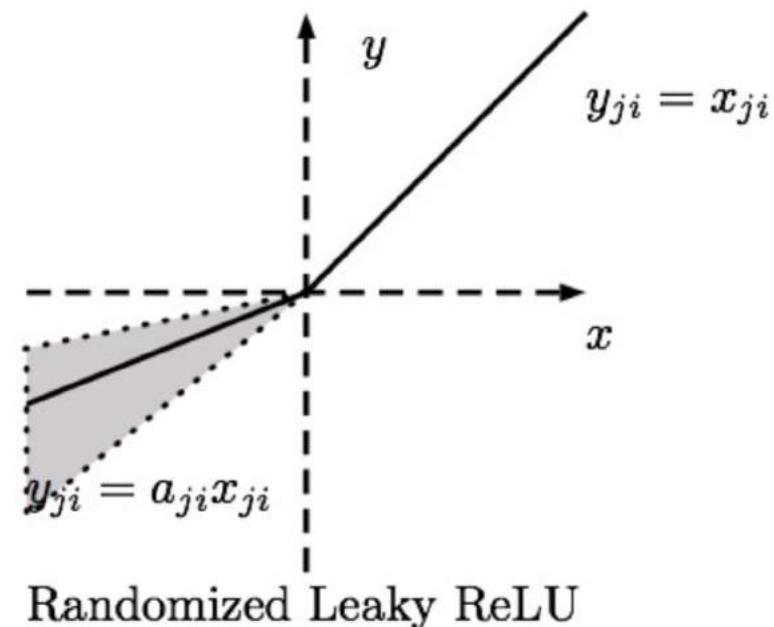
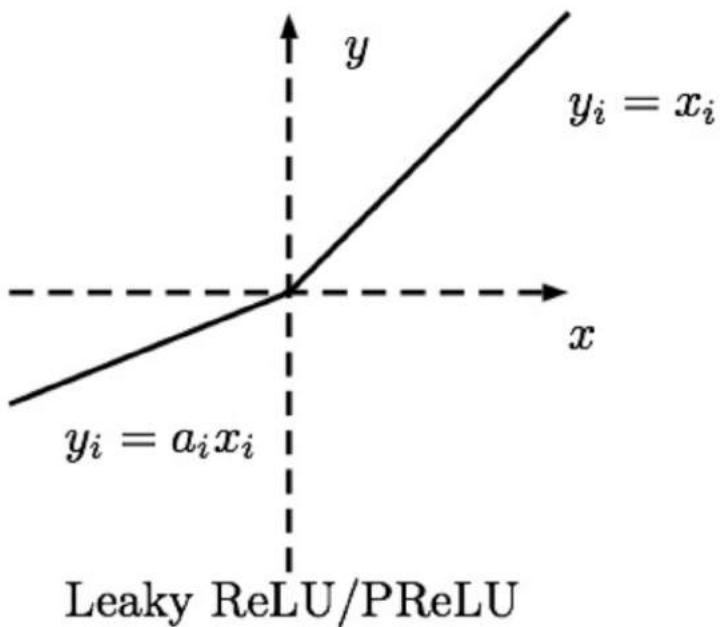
Нелинейности: \tanh



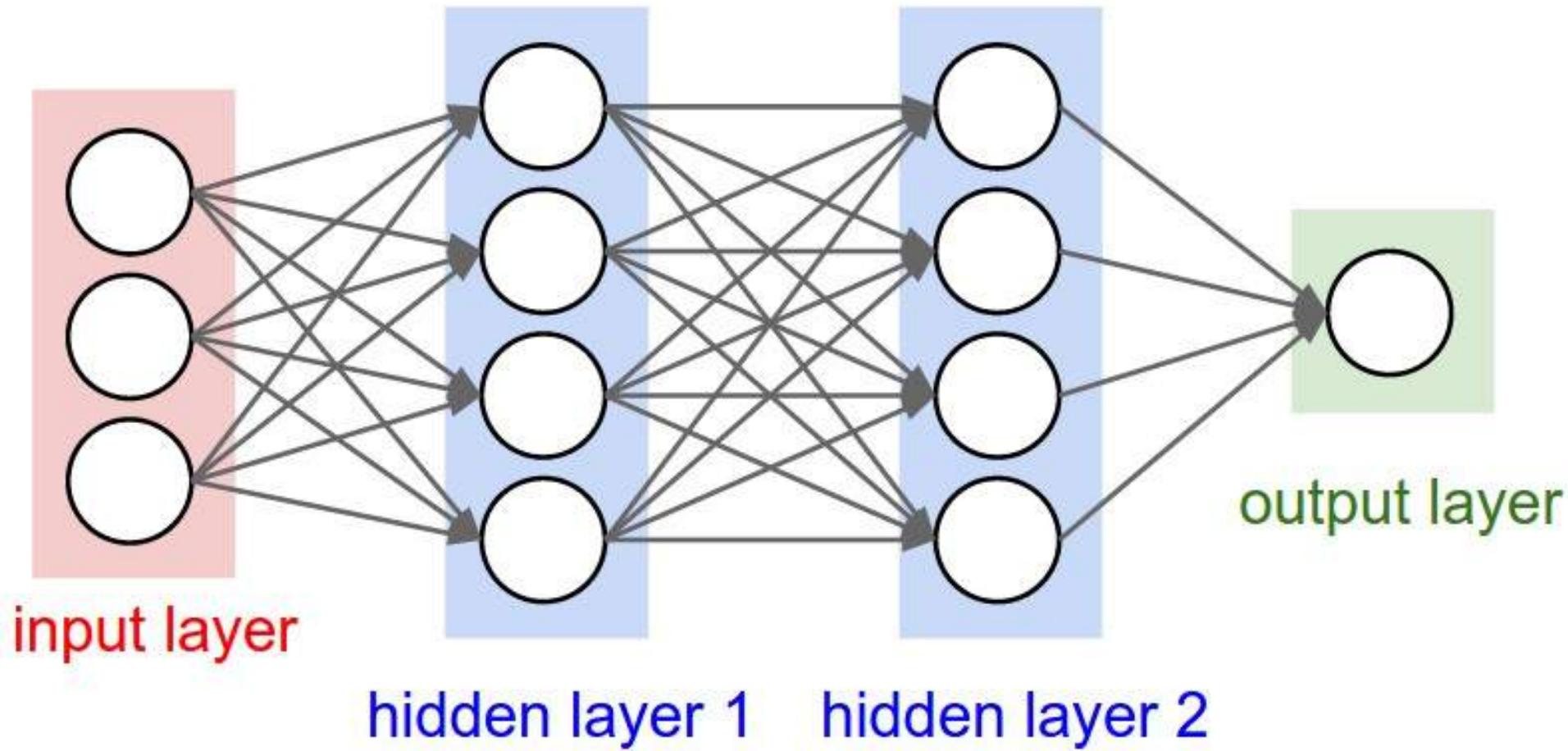
Нелинейности: ReLU



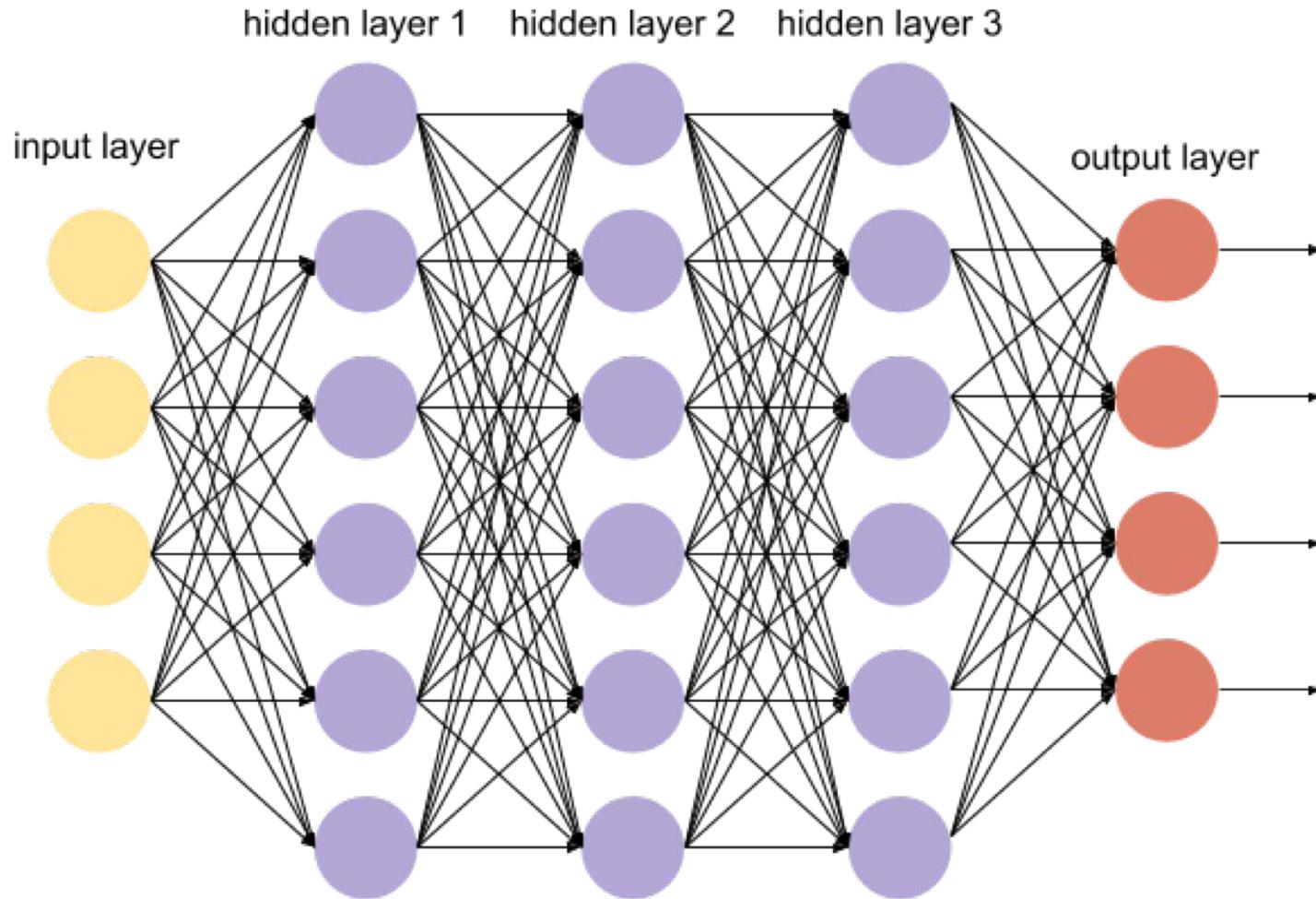
Нелинейности: Leaky ReLU



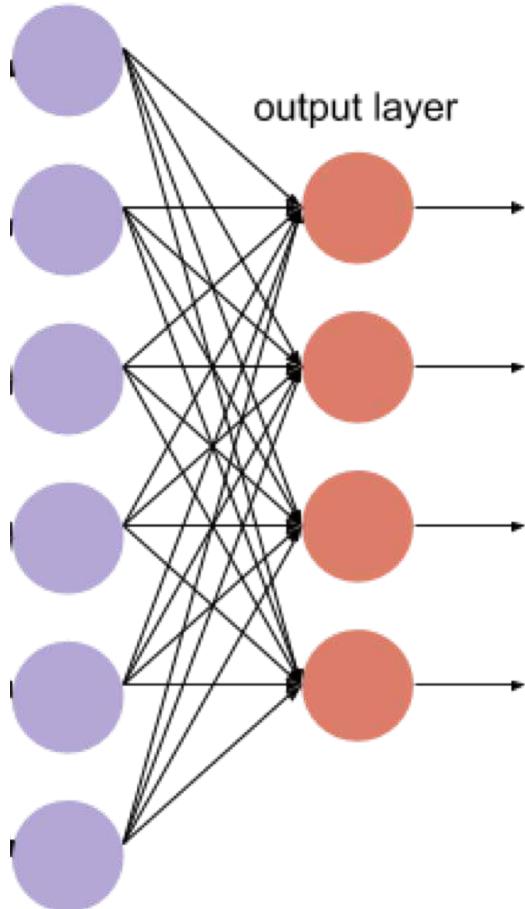
Нейронная сеть (один выход)



Нейронная сеть (много выходов)

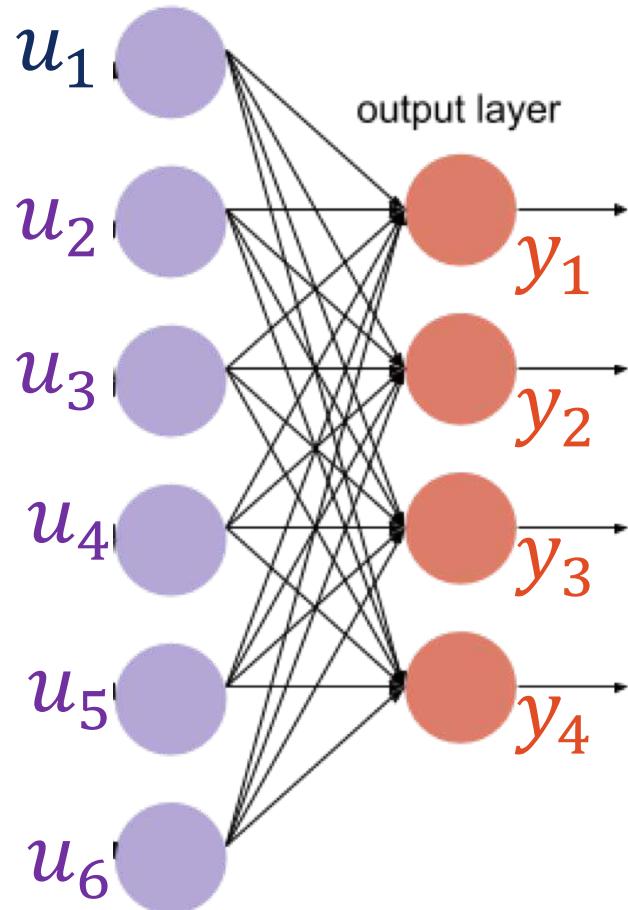


Еще один пример нелинейности: Softmax



Как из чисел в фиолетовых нейронах
получить вероятности классов в
красных?

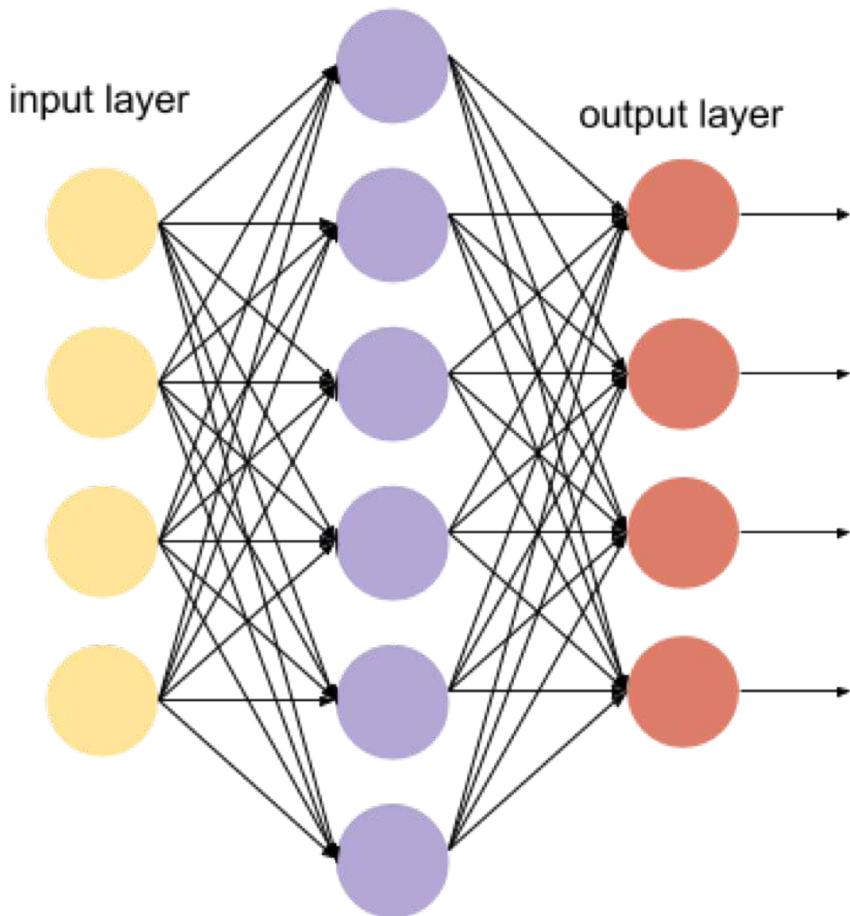
Softmax



Как из чисел в фиолетовых нейронах
получить вероятности классов в
красных?

$$y_k = \frac{e^{u_k}}{\sum_i e^{u_i}}$$

Универсальная теорема аппроксимации



Капелька оптимизма перед обсуждением обучения:
В 1989 г. Джорджем Цыбенко (George Cybenko) была доказана Universal Approximation Theorem

Нестрого:

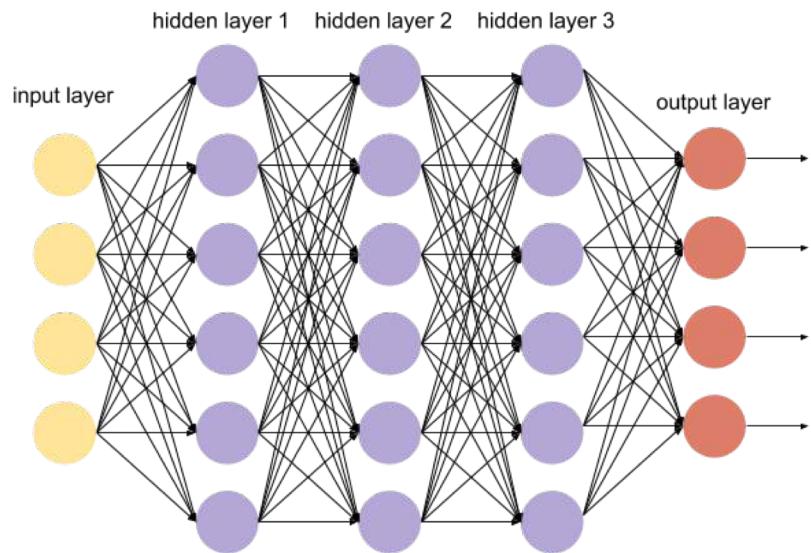
Если нам дана функция f и сказано, с какой точностью ее нужно приблизить (какой бы эта точность ни была) – мы всегда справимся с задачей даже однослоиной нейросетью, т.е. сможем подобрать подходящее количество нейронов и веса

Чуть более строго (для математиков):

f должна быть непрерывна на некотором компакте в R^n и условия теоремы выполняются на нём же

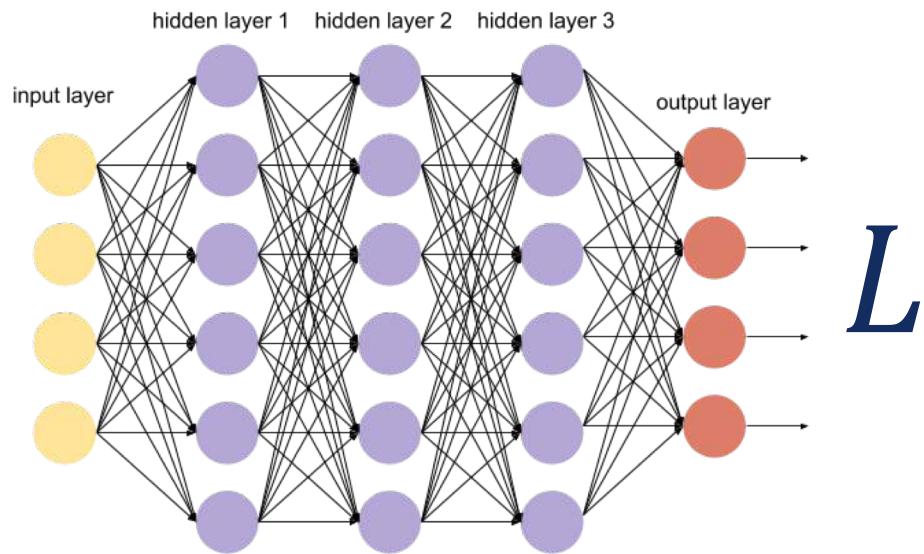
2. Обучение нейросетей

Обучение сети



Задача обучения – настроить веса связей между нейронами на основе обучающей выборки

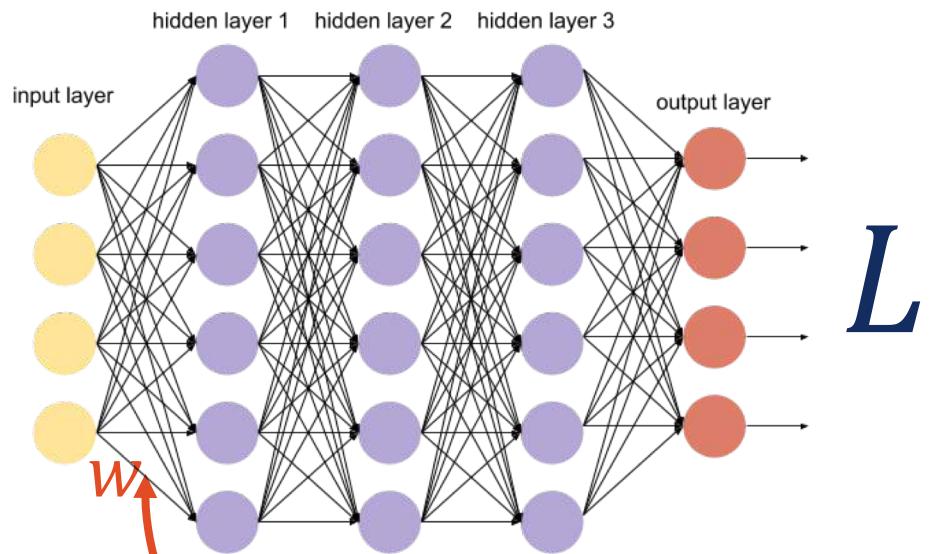
Обучение сети



Задача обучения – настроить веса связей между нейронами на основе обучающей выборки

1. Выбираем функцию потерь и записываем ошибку сети

Обучение сети

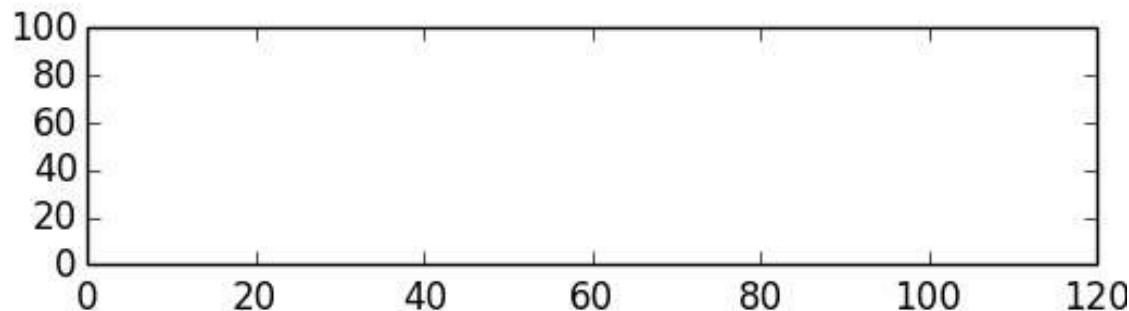
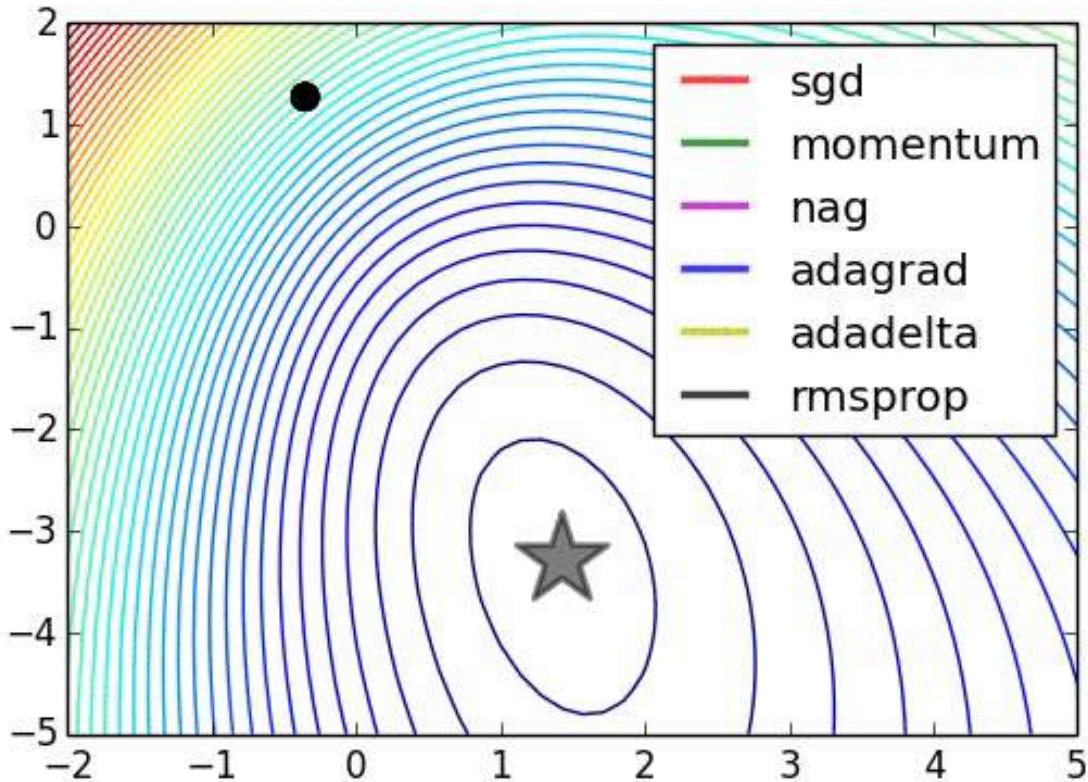


$$w^{(t+1)} = w^{(t)} - \gamma_t \frac{\partial L}{\partial w}$$

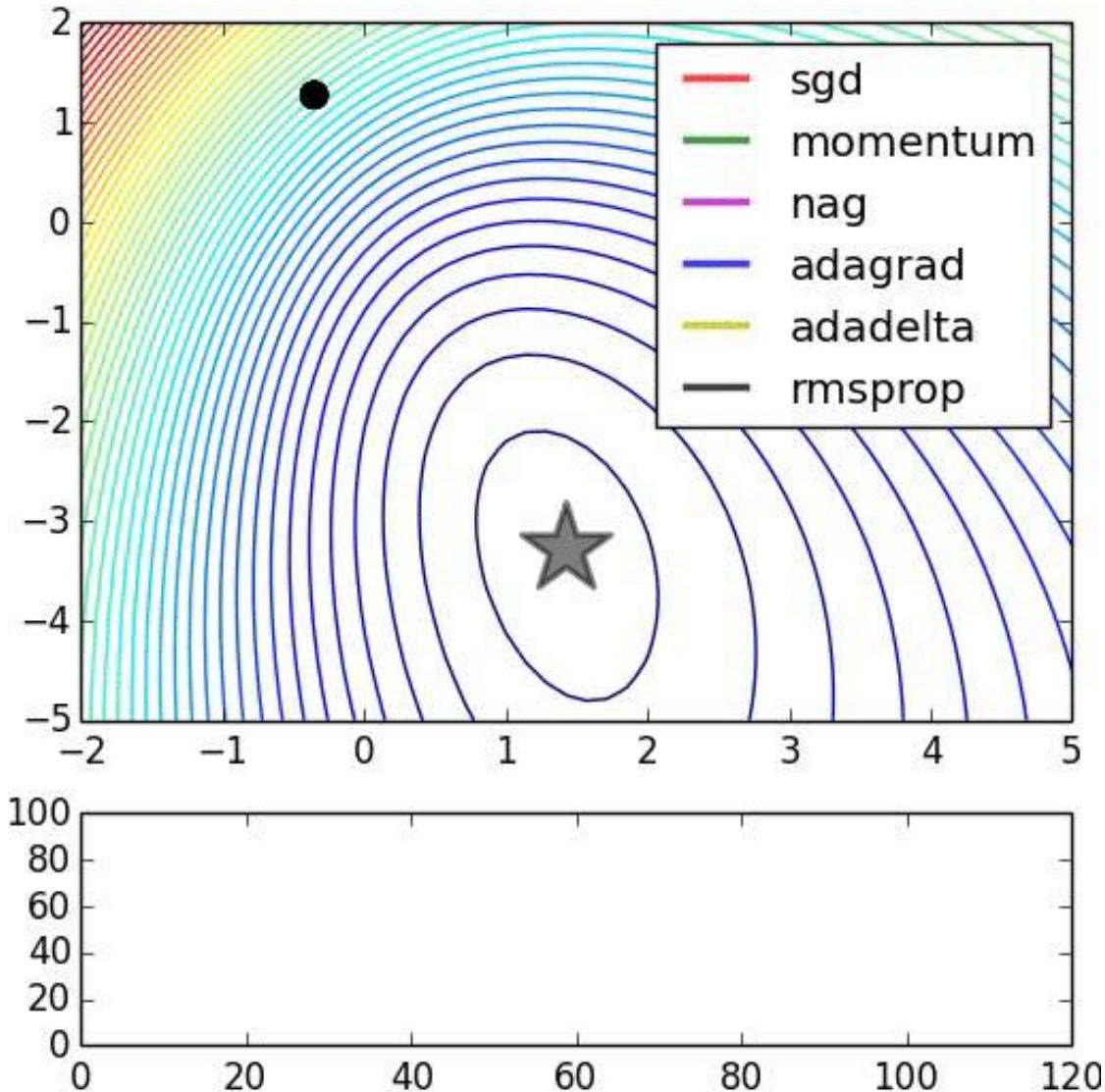
Задача обучения – настроить веса связей между нейронами на основе обучающей выборки

1. Выбираем функцию потерь и записываем ошибку сети
2. Обучаем веса с помощью SGD

Модификации SGD



Модификации SGD



Momentum:

Запоминает направление предыдущего шага и с небольшим весом учитывает его в текущем (движение «по инерции»)

NAG (Nesterov Accelerated Gradient):

Модификация Momentum: антиградиент берем в той точке, куда шагнули бы по инерции

Adagrad, Adadelta, RMSprop, Adam:

Добавлены эвристики для настройки разного шага по разным координатам (весам)

Модификации SGD



Momentum:

Запоминает направление предыдущего шага и с небольшим весом учитывает его в текущем (движение «по инерции»)

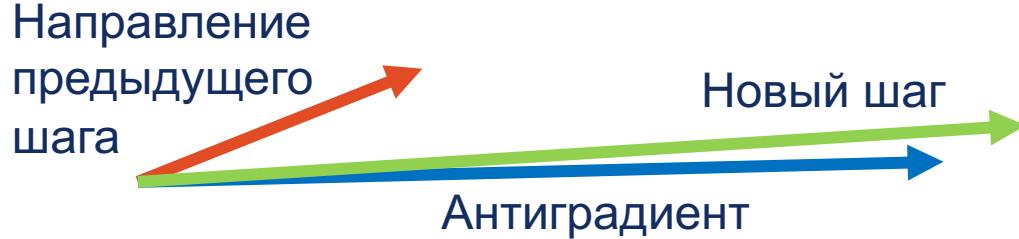
Модификации SGD



Momentum:

Запоминает направление предыдущего шага и с небольшим весом учитывает его в текущем (движение «по инерции»)

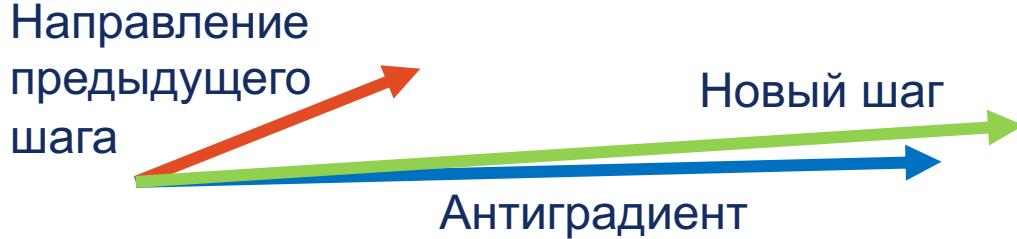
Модификации SGD



Momentum:

Запоминает направление предыдущего шага и с небольшим весом учитывает его в текущем (движение «по инерции»)

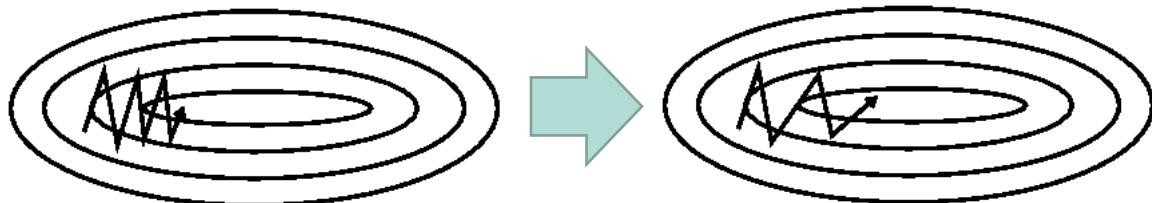
Модификации SGD



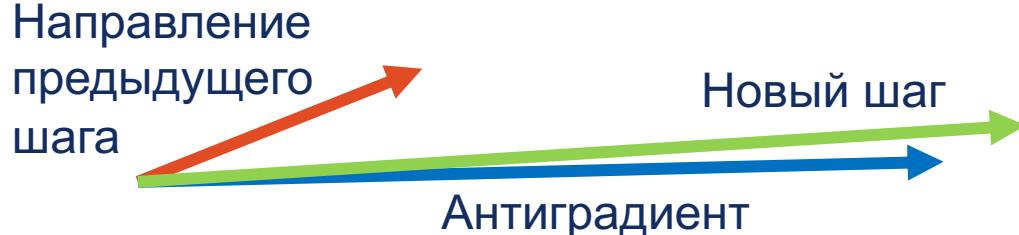
Momentum:

Запоминает направление предыдущего шага и с небольшим весом учитывает его в текущем (движение «по инерции»)

Идея Momentum:



Модификации SGD



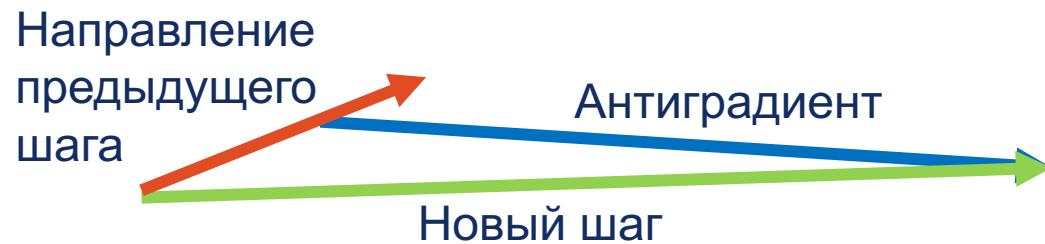
Momentum:

Запоминает направление предыдущего шага и с небольшим весом учитывает его в текущем (движение «по инерции»)

NAG (Nesterov Accelerated Gradient):

Модификация Momentum: антиградиент берем в той точке, куда шагнули бы по инерции

Модификации SGD



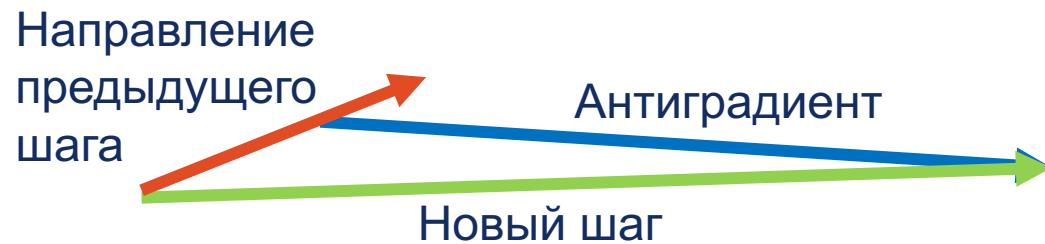
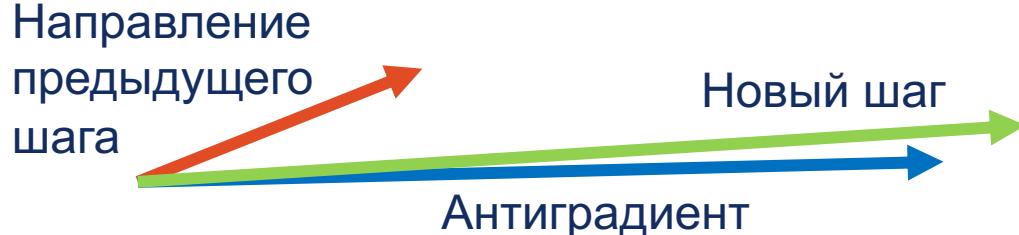
Momentum:

Запоминает направление предыдущего шага и с небольшим весом учитывает его в текущем (движение «по инерции»)

NAG (Nesterov Accelerated Gradient):

Модификация Momentum: антиградиент берем в той точке, куда шагнули бы по инерции

Модификации SGD



$$w^{(t+1)} = w^{(t)} - \gamma_t \frac{\partial L}{\partial w}$$

Momentum:

Запоминает направление предыдущего шага и с небольшим весом учитывает его в текущем (движение «по инерции»)

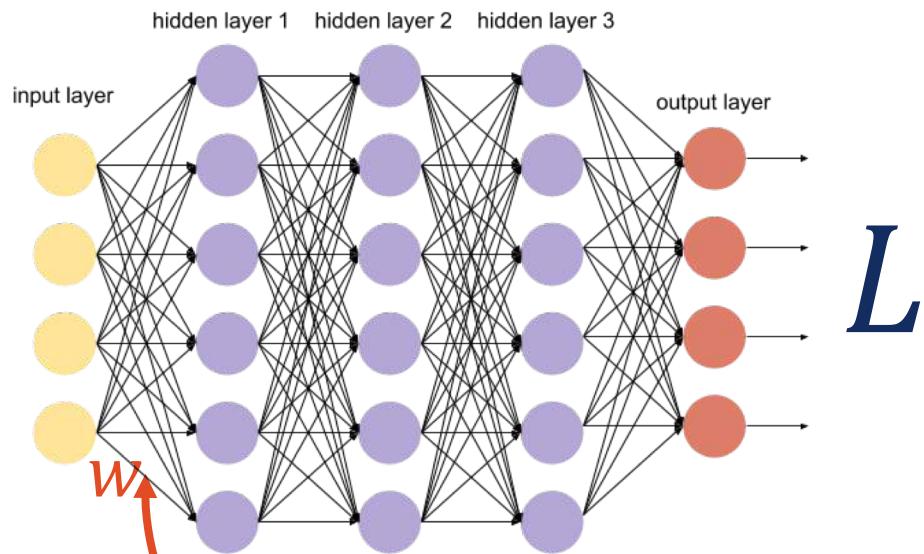
NAG (Nesterov Accelerated Gradient):

Модификация Momentum: антиградиент берем в той точке, куда шагнули бы по инерции

Adagrad, Adadelta, RMSprop, Adam:

Добавлены эвристики для настройки разного шага по разным координатам (весам)

Обучение сети



L

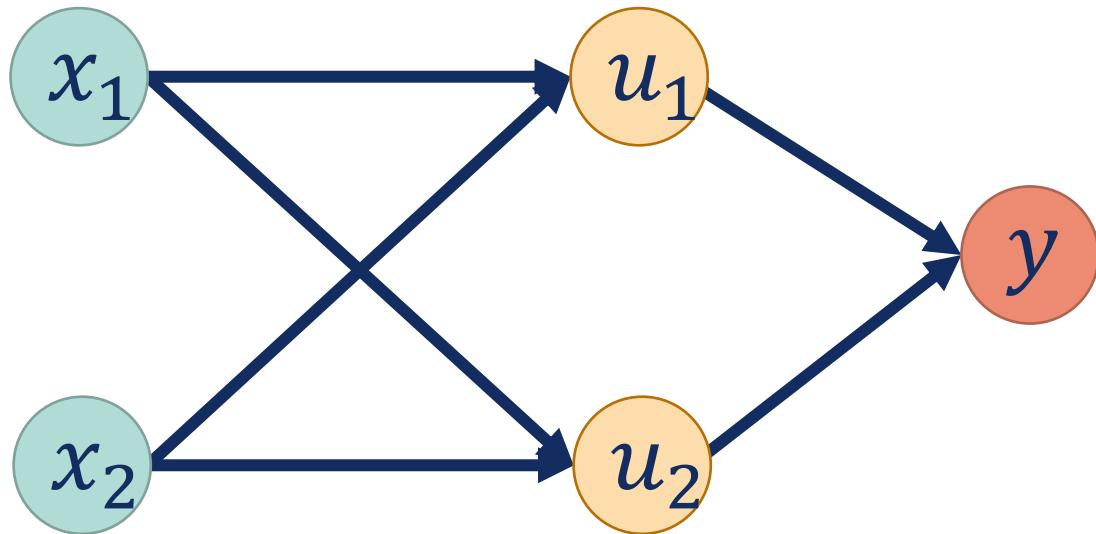
$$w^{(t+1)} = w^{(t)} - \gamma_t \frac{\partial L}{\partial w}$$

Задача обучения – настроить веса связей между нейронами на основе обучающей выборки

1. Выбираем функцию потерь и записываем ошибку сети
2. Обучаем веса с помощью SGD

Проблема: не выписывать же нам все производные аналитически?!

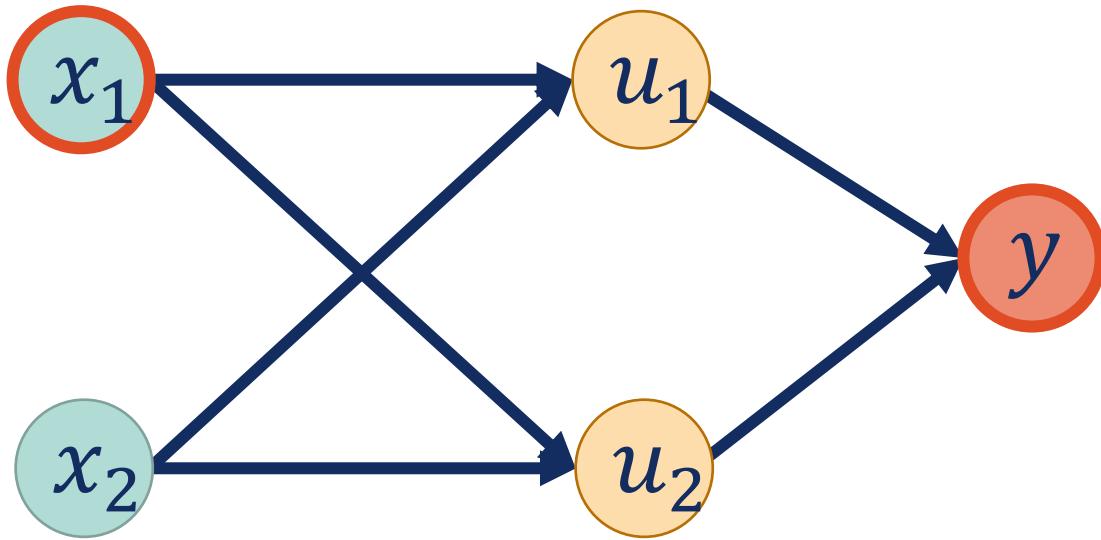
Граф вычислений



Пример для случая:

Переменная y вычисляется по переменным u_1 и u_2 , которые в свою очередь вычисляются по x_1 и x_2

Граф вычислений и chain rule

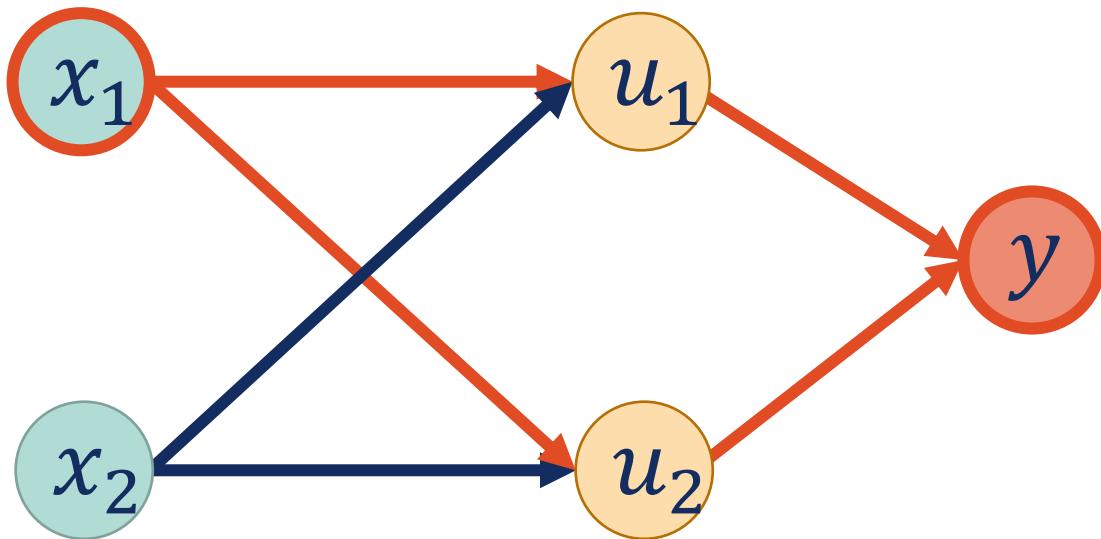


Как вычислить $\frac{\partial y}{\partial x_1}$?

Пример для случая:

Переменная y вычисляется по переменным u_1 и u_2 , которые в свою очередь вычисляются по x_1 и x_2

Граф вычислений и chain rule



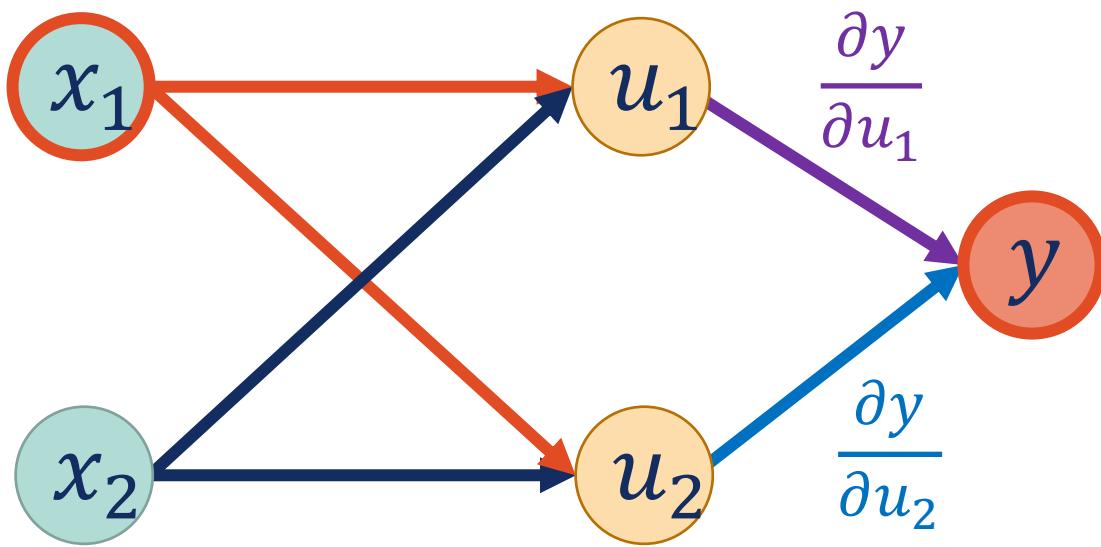
Как вычислить $\frac{\partial y}{\partial x_1}$?

$$\frac{\partial y}{\partial x_1} = \frac{\partial y}{\partial u_1} \frac{\partial u_1}{\partial x_1} + \frac{\partial y}{\partial u_2} \frac{\partial u_2}{\partial x_1}$$

Пример для случая:

Переменная y вычисляется по переменным u_1 и u_2 , которые в свою очередь вычисляются по x_1 и x_2

Граф вычислений и chain rule



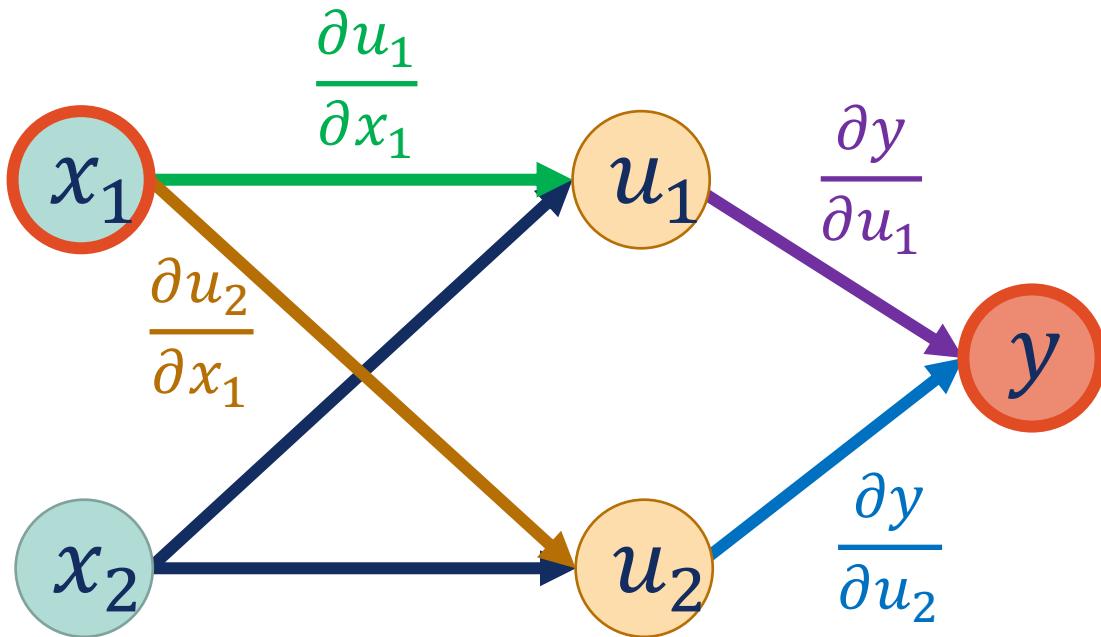
Как вычислить $\frac{\partial y}{\partial x_1}$?

$$\frac{\partial y}{\partial x_1} = \frac{\partial y}{\partial u_1} \frac{\partial u_1}{\partial x_1} + \frac{\partial y}{\partial u_2} \frac{\partial u_2}{\partial x_1}$$

Пример для случая:

Переменная y вычисляется по переменным u_1 и u_2 , которые в свою очередь вычисляются по x_1 и x_2

Граф вычислений и chain rule



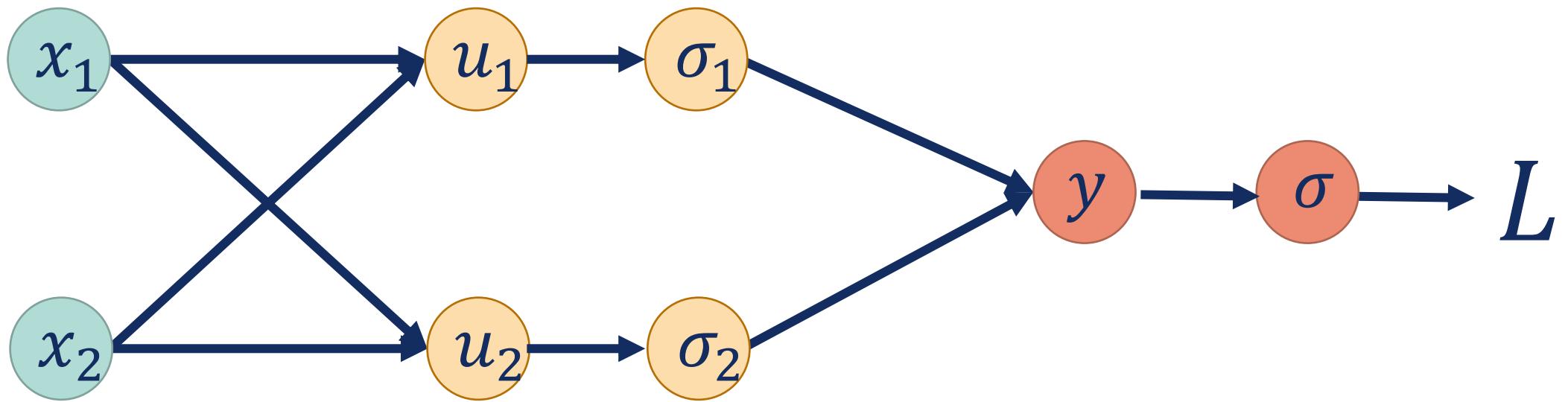
Как вычислить $\frac{\partial y}{\partial x_1}$?

$$\frac{\partial y}{\partial x_1} = \frac{\partial y}{\partial u_1} \frac{\partial u_1}{\partial x_1} + \frac{\partial y}{\partial u_2} \frac{\partial u_2}{\partial x_1}$$

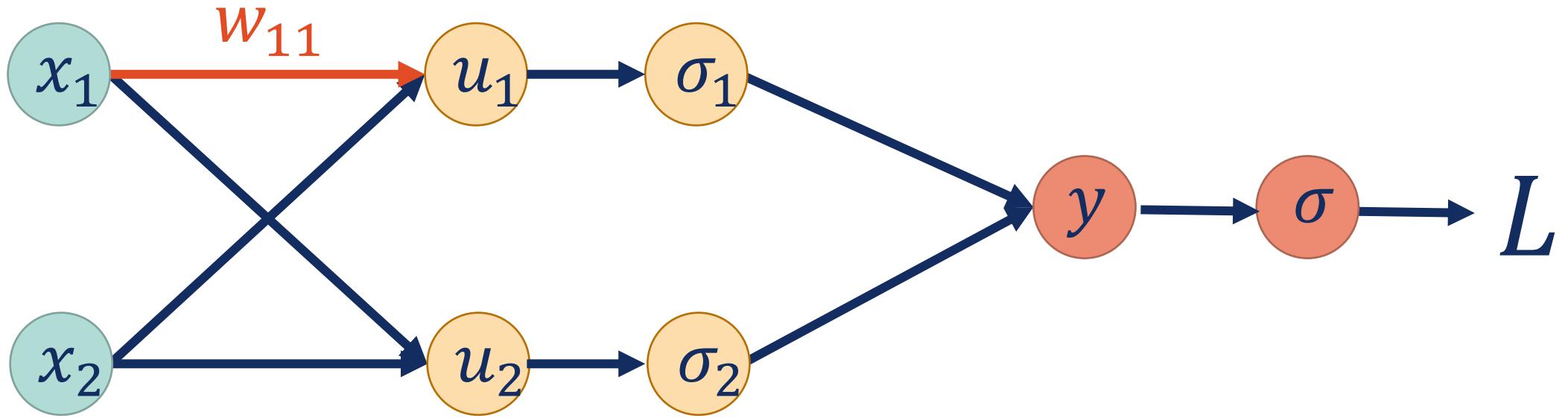
Пример для случая:

Переменная y вычисляется по переменным u_1 и u_2 , которые в свою очередь вычисляются по x_1 и x_2

Граф вычислений для нейросети

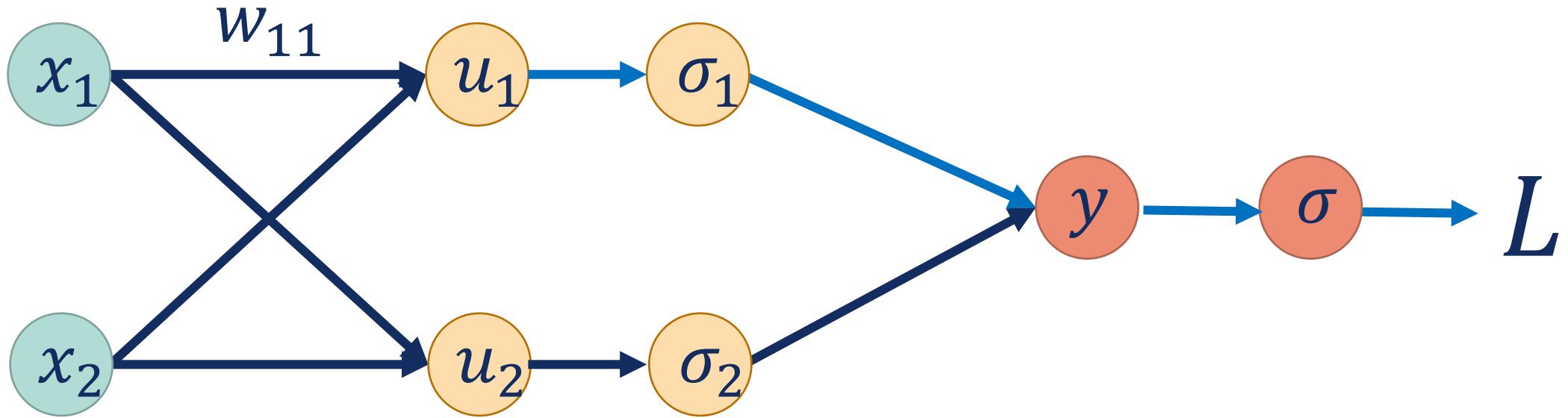


Граф вычислений для нейросети



$$w_{11}^{(t+1)} = w_{11}^{(t)} - \gamma_t \frac{\partial L}{\partial w_{11}}$$

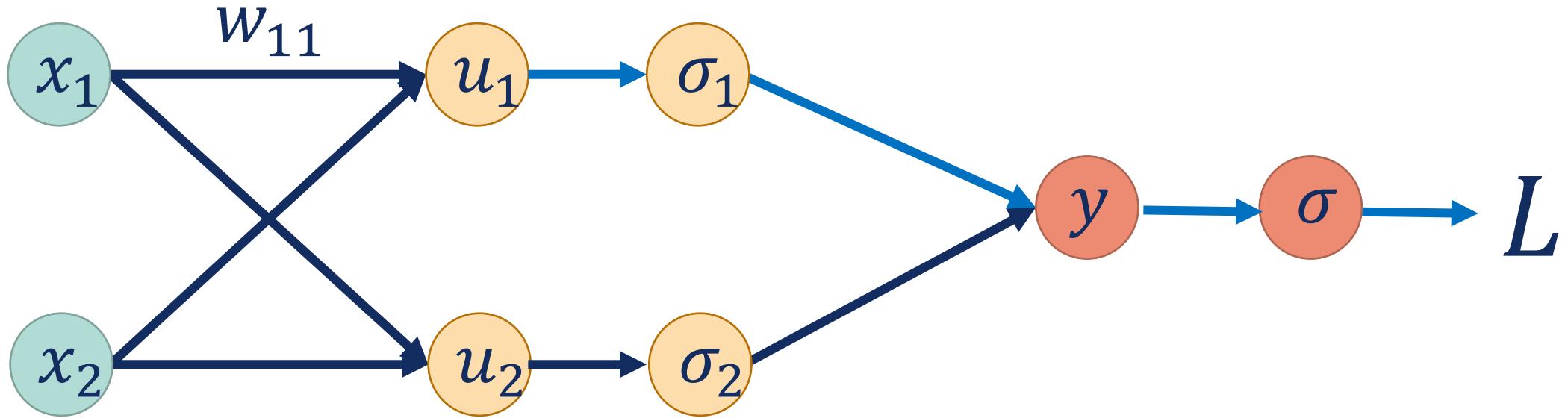
Граф вычислений для нейросети



$$w_{11}^{(t+1)} = w_{11}^{(t)} - \gamma_t \frac{\partial L}{\partial w_{11}}$$

$$\frac{\partial L}{\partial w_{11}} = \frac{\partial L}{\partial u_1} \frac{\partial u_1}{\partial w_{11}} = \frac{\partial L}{\partial u_1} x_1$$

Граф вычислений для нейросети



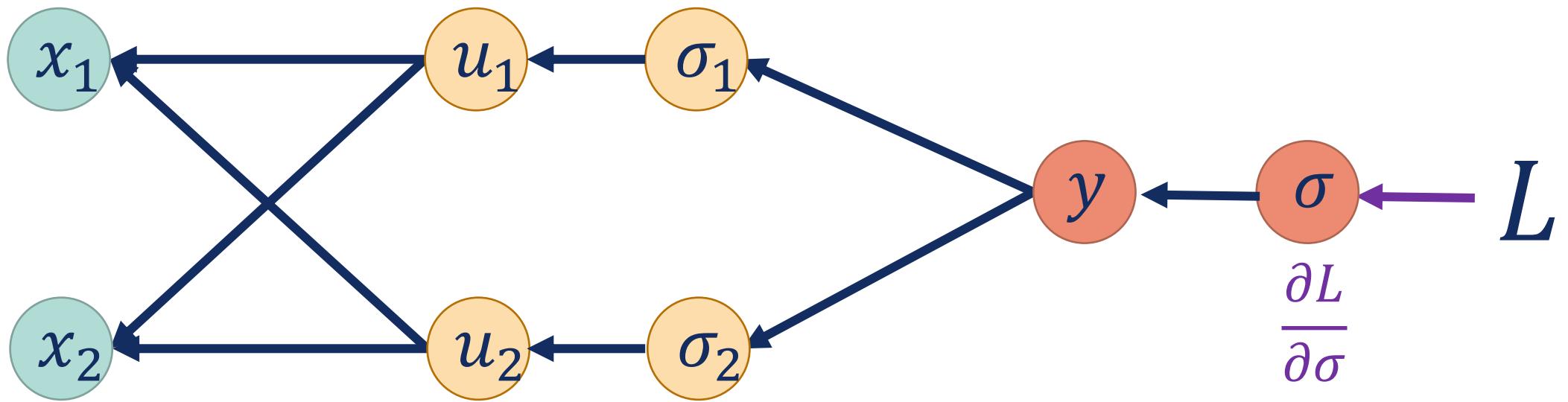
$$w_{11}^{(t+1)} = w_{11}^{(t)} - \gamma_t \frac{\partial L}{\partial w_{11}}$$

$$\frac{\partial L}{\partial w_{11}} = \frac{\partial L}{\partial u_1} \frac{\partial u_1}{\partial w_{11}} = \frac{\partial L}{\partial u_1} x_1$$

Вывод: для обучения нужны производные L по выходам всех нейронов

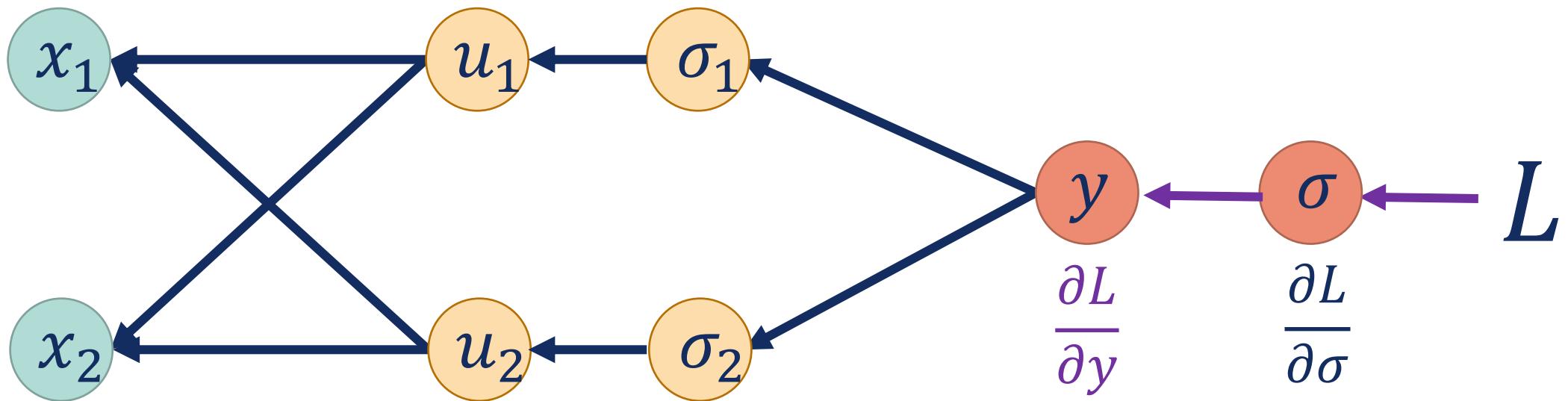
Backpropagation

Backprop – эффективный способ посчитать производные L по нейронам:



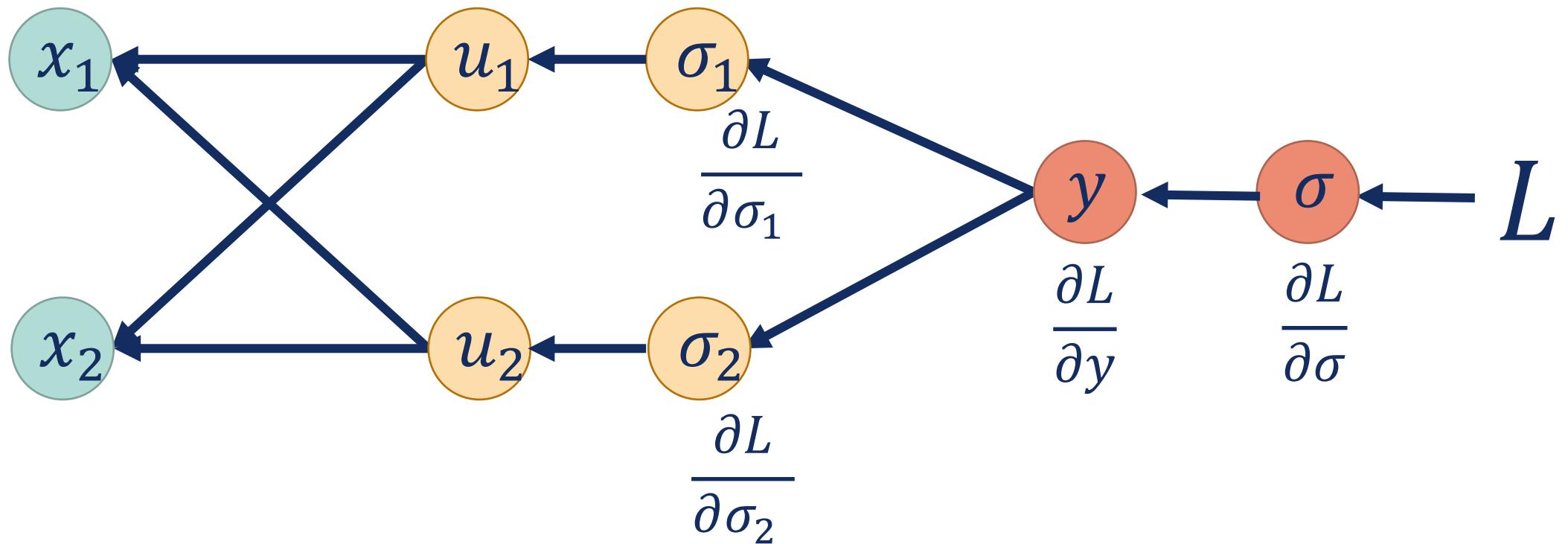
Backpropagation

Backprop – эффективный способ посчитать производные L по нейронам:



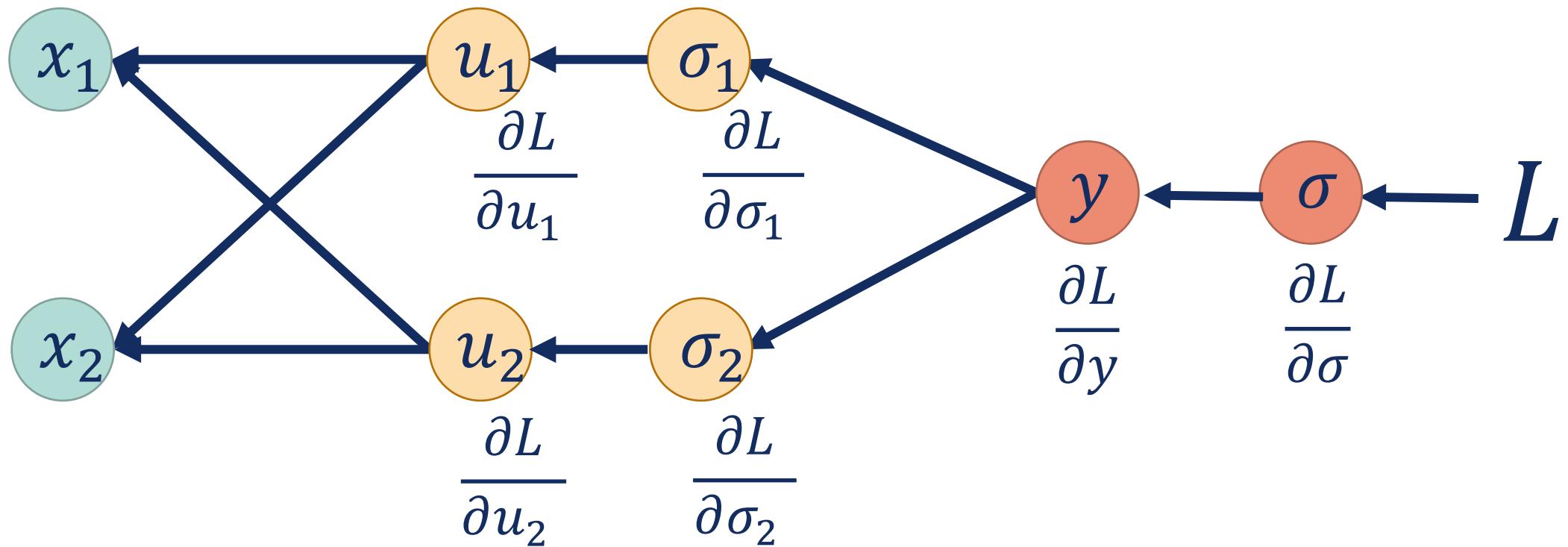
Backpropagation

Backprop – эффективный способ посчитать производные L по нейронам:



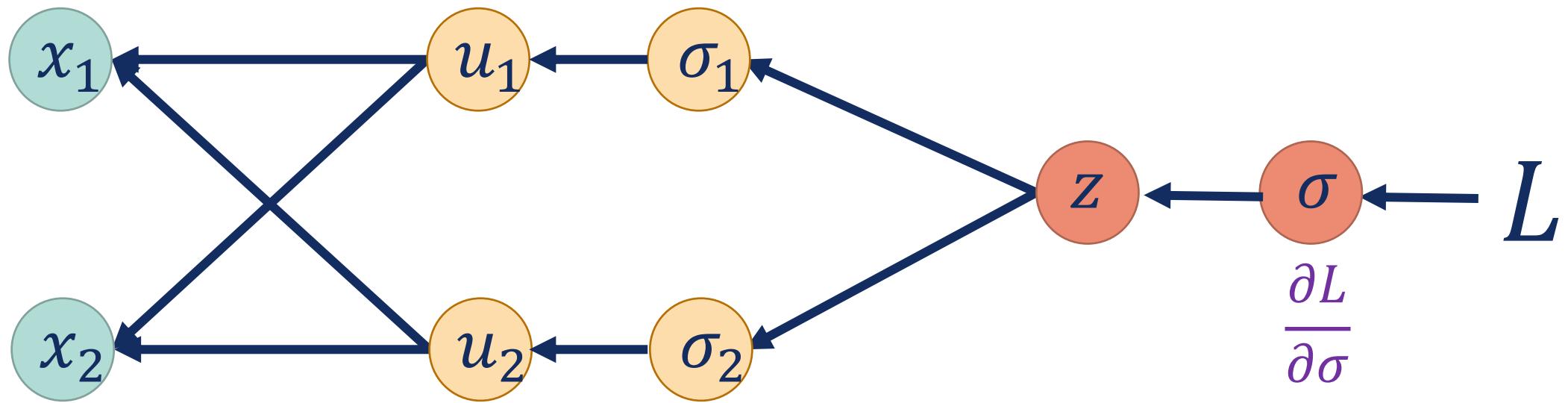
Backpropagation

Backprop – эффективный способ посчитать производные L по нейронам:



Другое название: Error Backpropagation

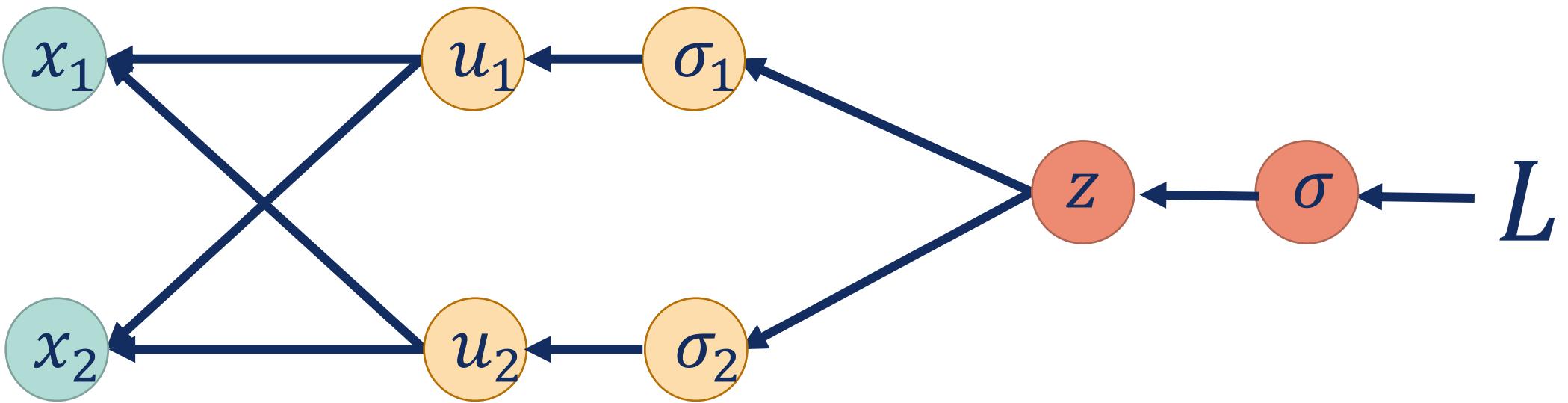
Раньше этот метод часто называли обратным распространением ошибки



Если $L = \frac{1}{2}(\sigma - y_{true})^2$,
 $\frac{\partial L}{\partial \sigma} = \sigma - y_{true}$ - ошибка прогноза

Другое название: Error Backpropagation

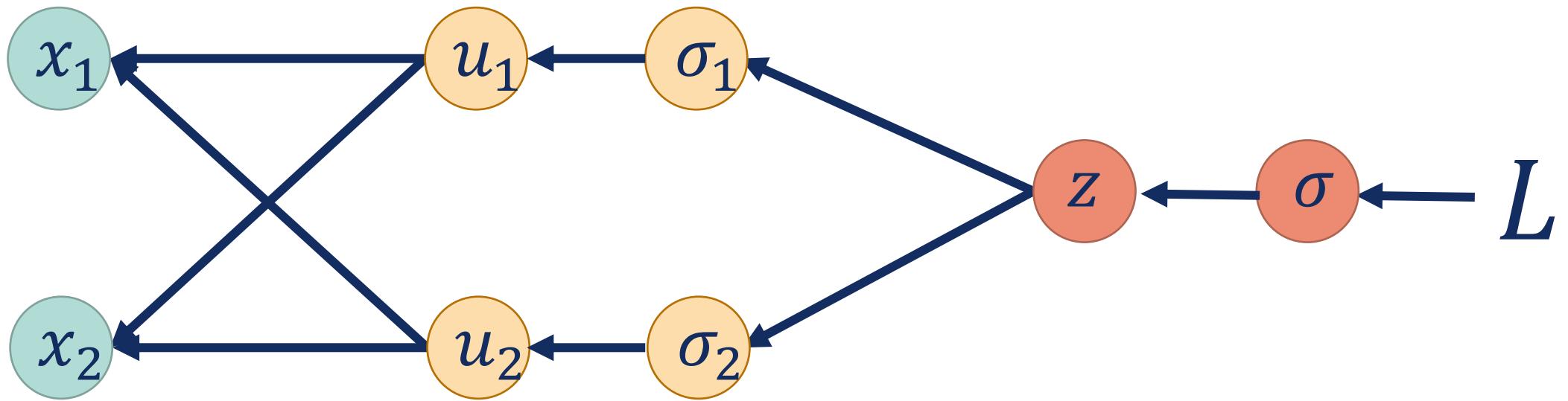
Раньше этот метод часто называли обратным распространением **ошибки**



При градиентном спуске мы бы «подправляли» u_2 на величину $\frac{\partial L}{\partial u_2}$ - значит эта производная – аналог «ошибки» в этом нейроне

Другое название: Error Backpropagation

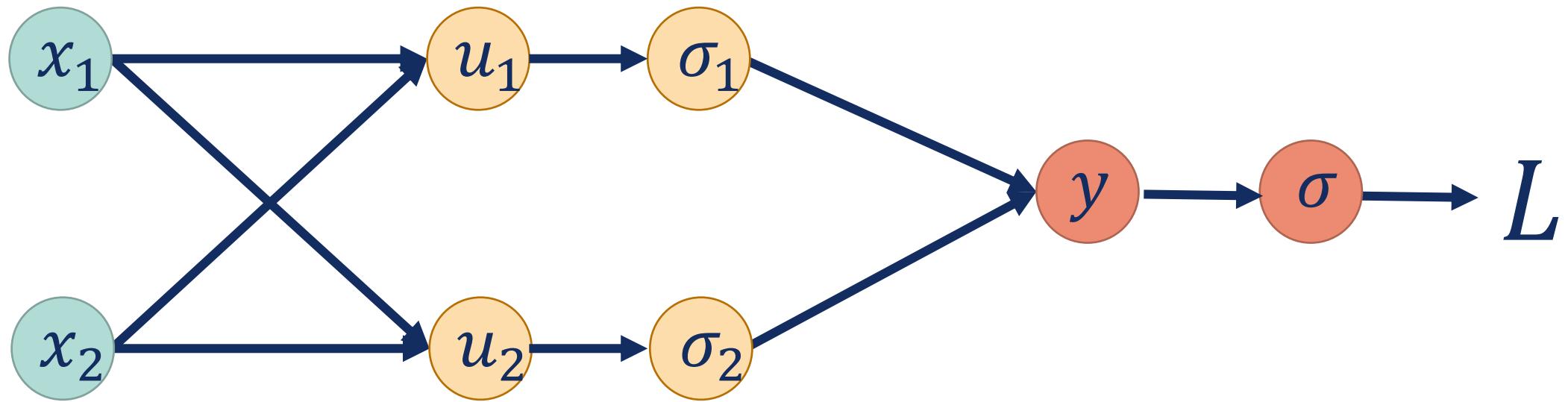
Раньше этот метод часто называли обратным распространением ошибки



Получается, мы вычисляем ошибку на выходе и «распространяем» ее в обратном направлении (ко входу), вычисляя «ошибки» во всех нейронах по пути

Backprop: как делать

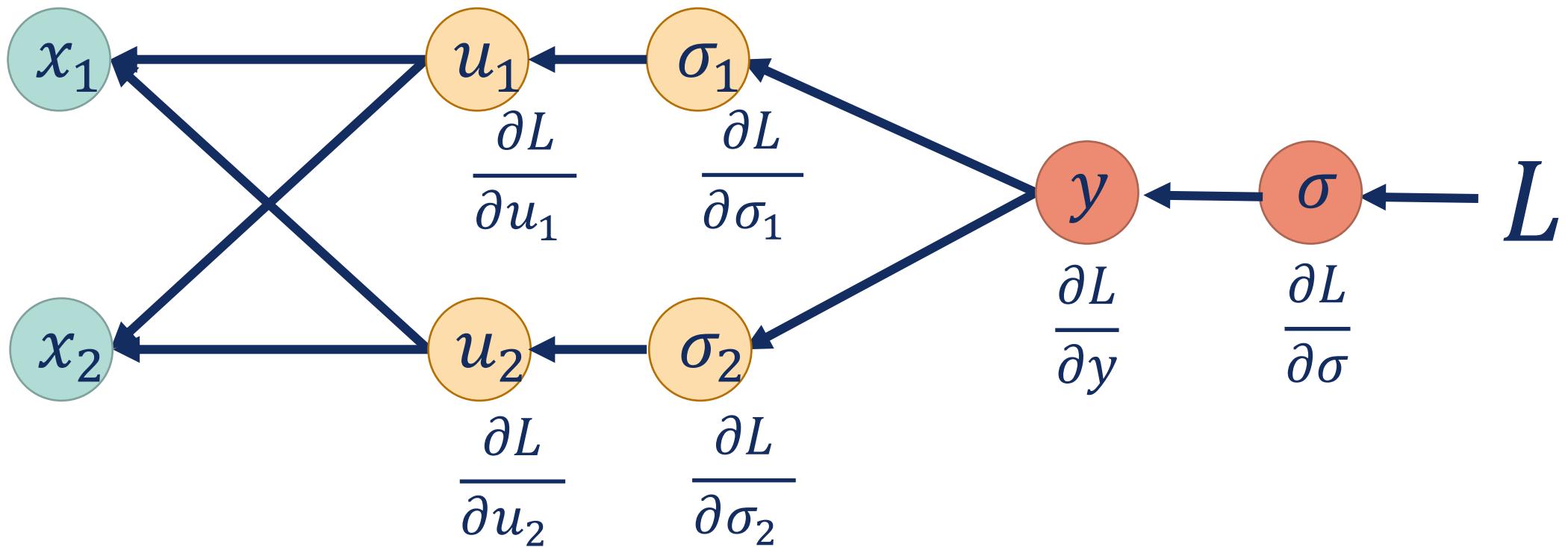
Чередуем **forward pass** (вычисление значений в нейронах)



Этот шаг нам нужен, чтобы знать, в каких точках считать производные

Backprop: как делать

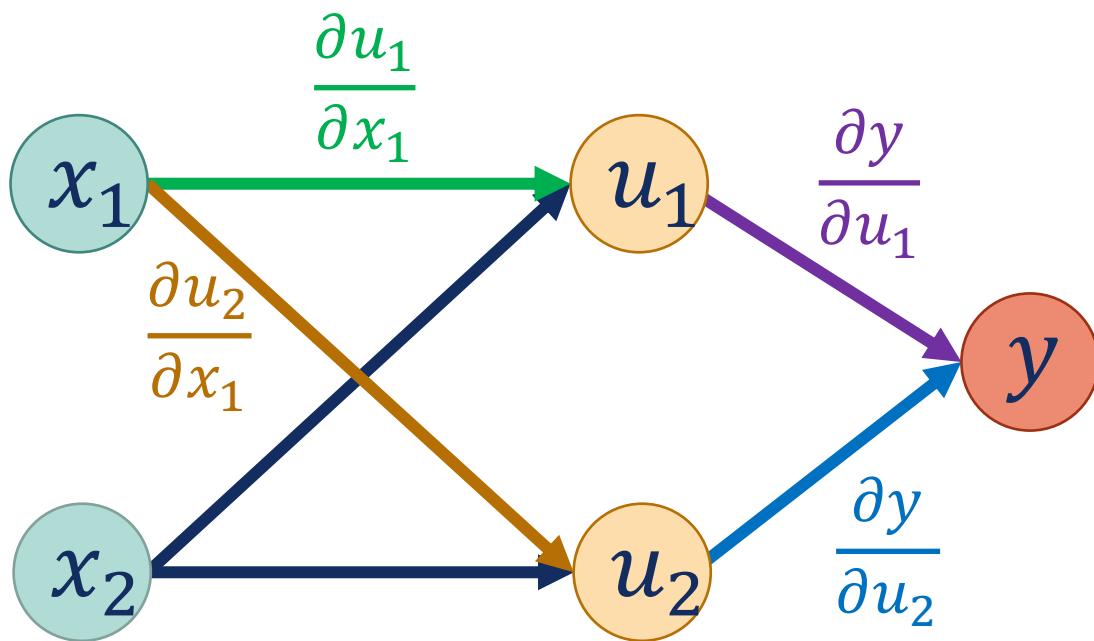
И **backward pass** (вычисление производных):



Как это реализовано в библиотеках

1. Для каждого типа слоя написан forward pass и backward pass
2. Операции оптимизированы за счет матричной записи и алгоритмов быстрых матричных вычислений (см. BLAS)

Инициализация весов



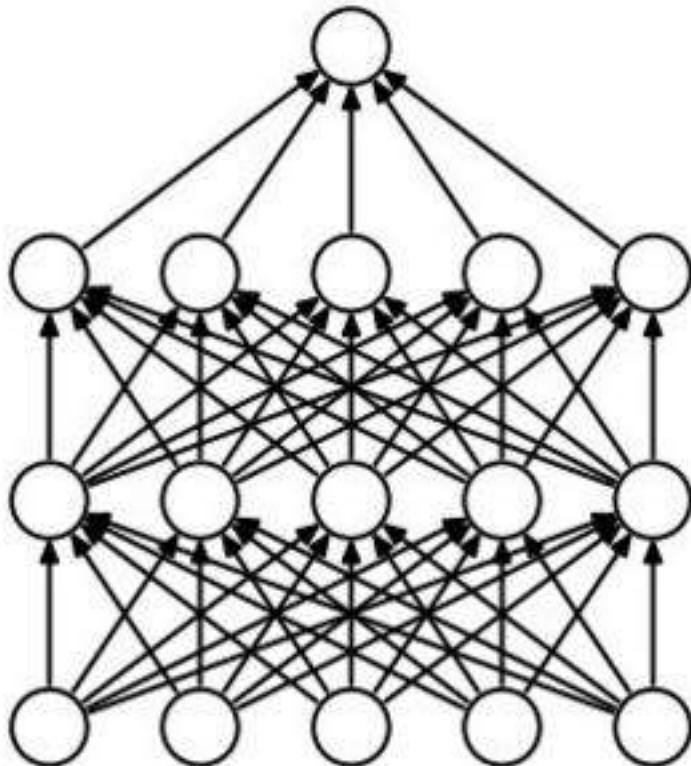
1. Все производные – либо веса между нейронами, либо производные нелинейностей, т.е. обновление весов пропорционально самим весам. Значит точно **не стоит брать все веса равными**
2. Инициализация из нормального распределения может приводить к затуханию или взрыву градиента (об этом позже) – но **есть эвристики для предотвращения эффекта***

* <https://medium.com/usf-msds/deep-learning-best-practices-1-weight-initialization-14e5c0295b94>

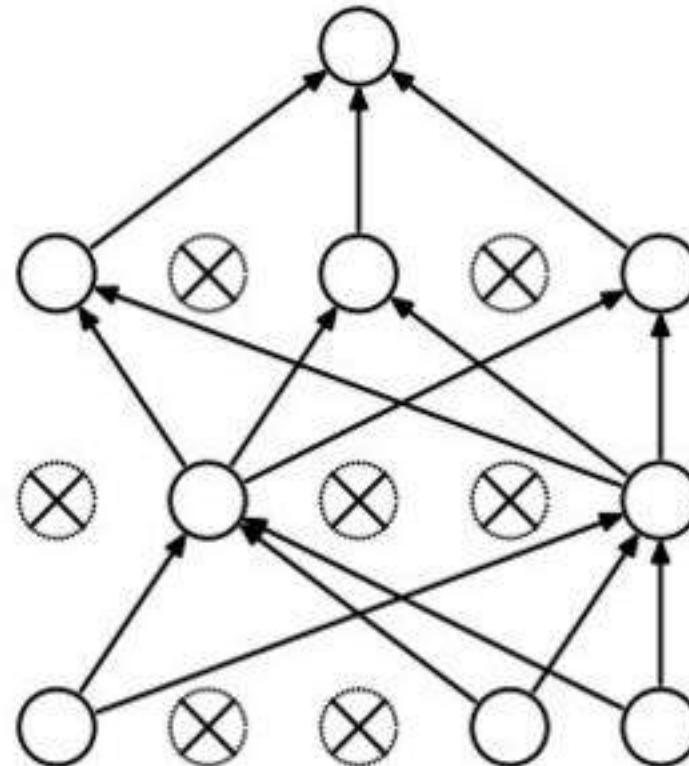
Проблемы backprop

1. Все проблемы SGD, в частности – застревание в острых локальных минимумах и легкое переобучение

Пример регуляризации: dropout



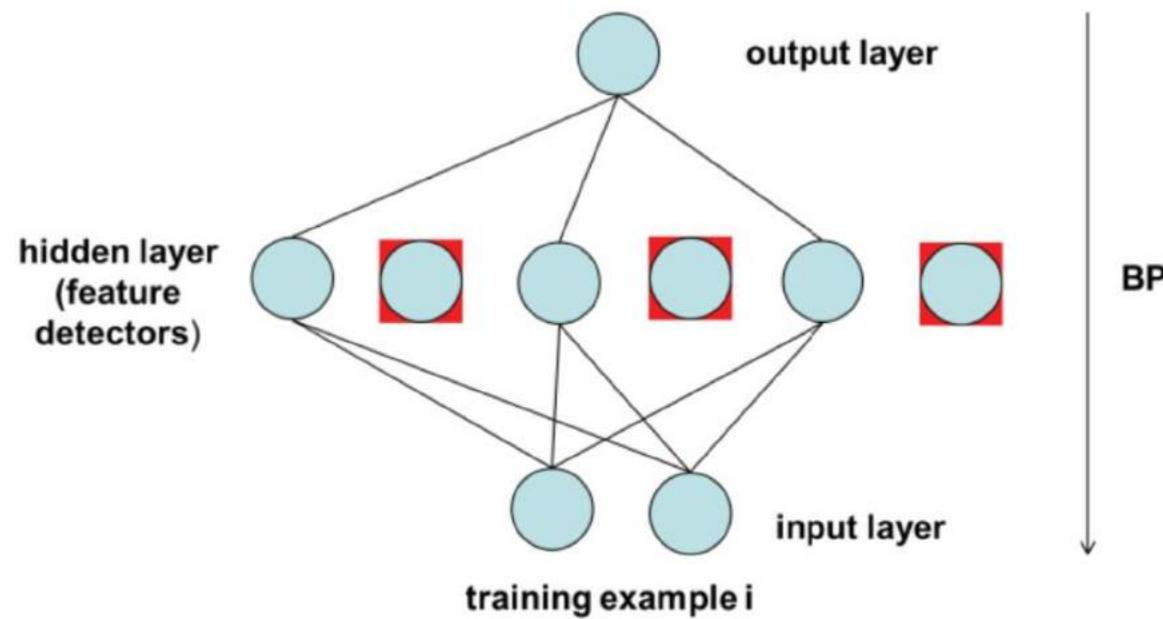
(a) Standard Neural Net



(b) After applying dropout.

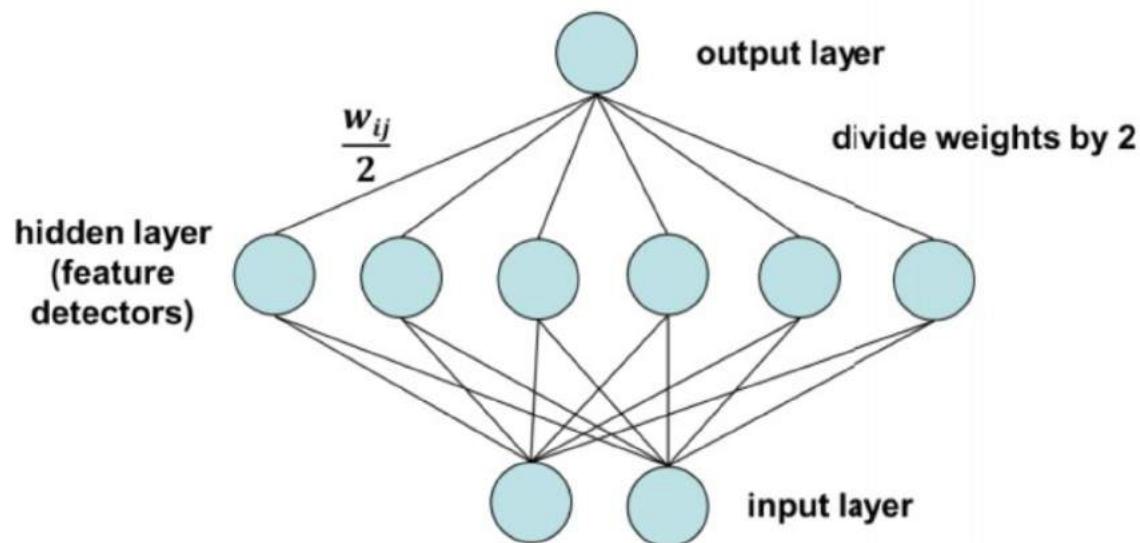
Dropout: обучение

С вероятностью r зануляем
выход каждого нейрона на слое



Dropout: применение

Домножаем выход каждого нейрона на $(1-p)$

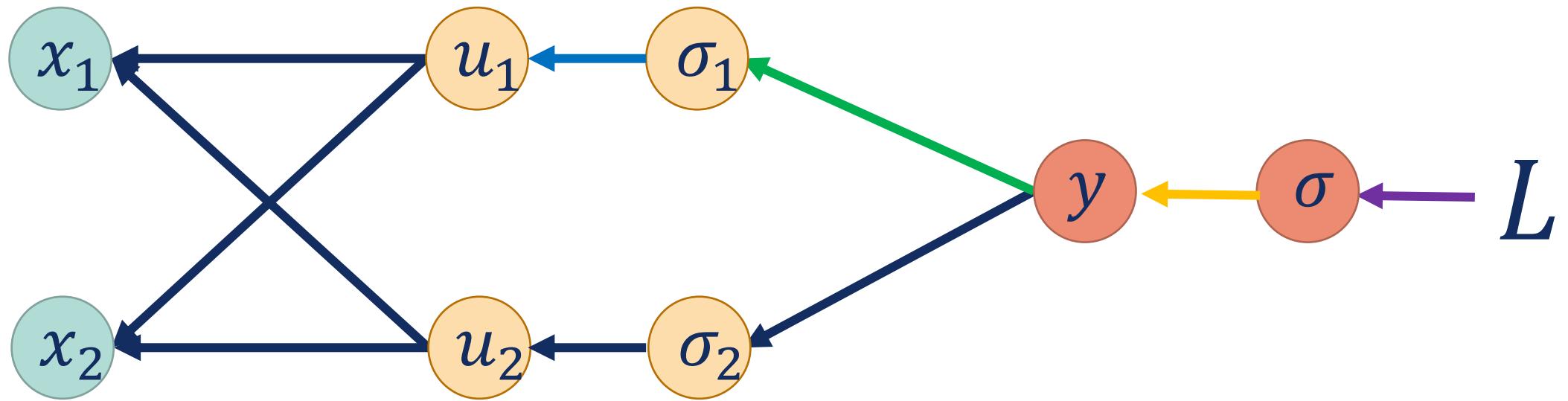


Проблемы backprop

1. Все проблемы SGD, в частности – застревание в локальных минимумах и легкое переобучение
2. Взрыв и затухание градиента (но на самом деле – это не совсем проблема backprop)

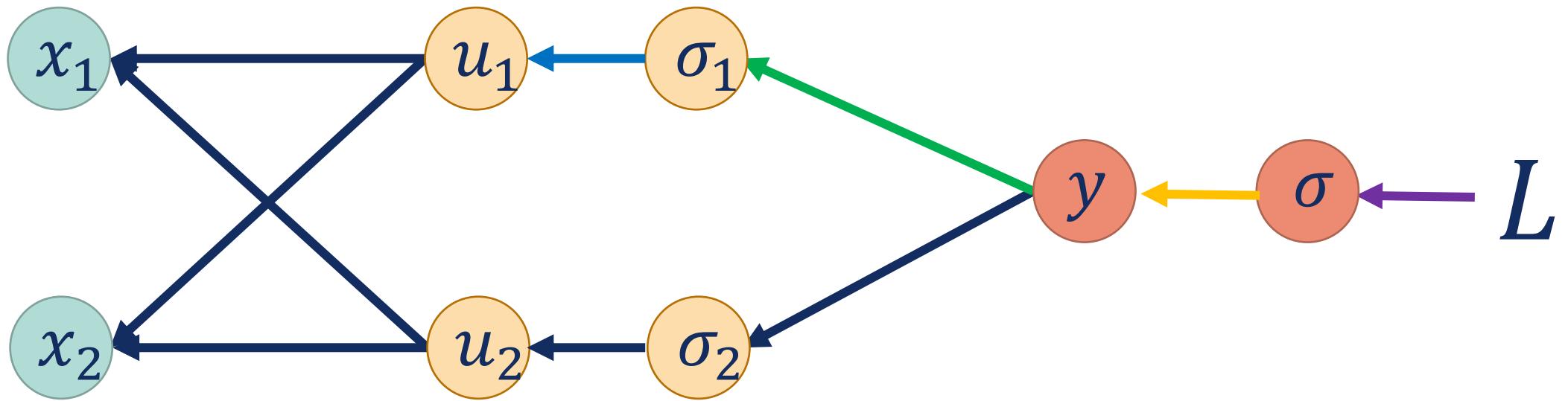
Затухание градиента (Gradient vanishing)

Производная по нейрону по chain rule получается из произведения производных по пути к нему от выхода



Затухание градиента (Gradient vanishing)

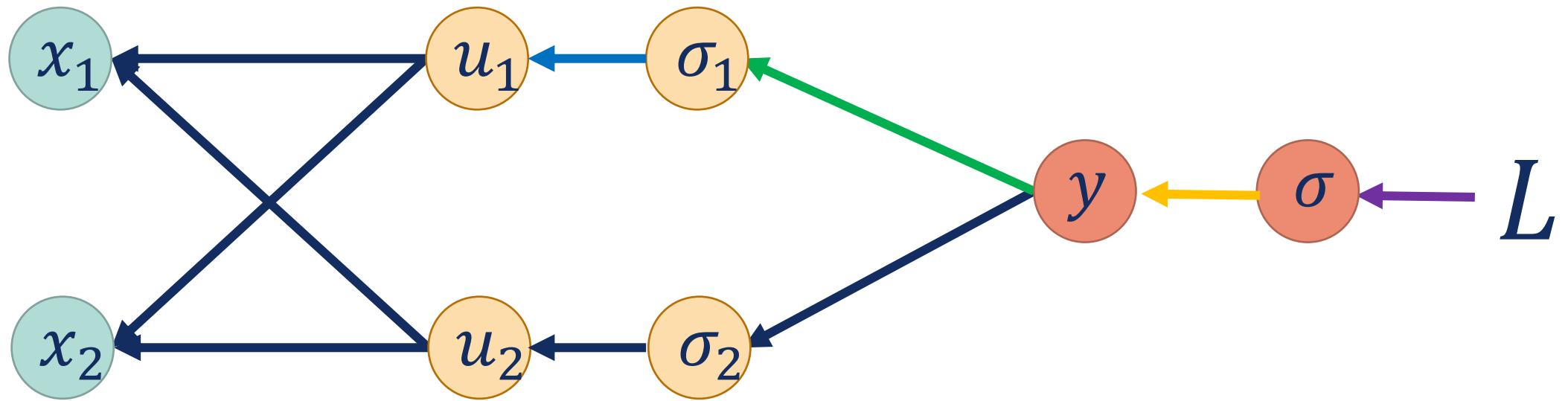
Производная по нейрону по chain rule получается из произведения производных по пути к нему от выхода



Если каждая из производных небольшая по модулю – произведение тоже будет маленьким. Чем больше слоев, тем меньше.

Затухание градиента (Gradient vanishing)

Производная по нейрону по chain rule получается из произведения производных по пути к нему от выхода

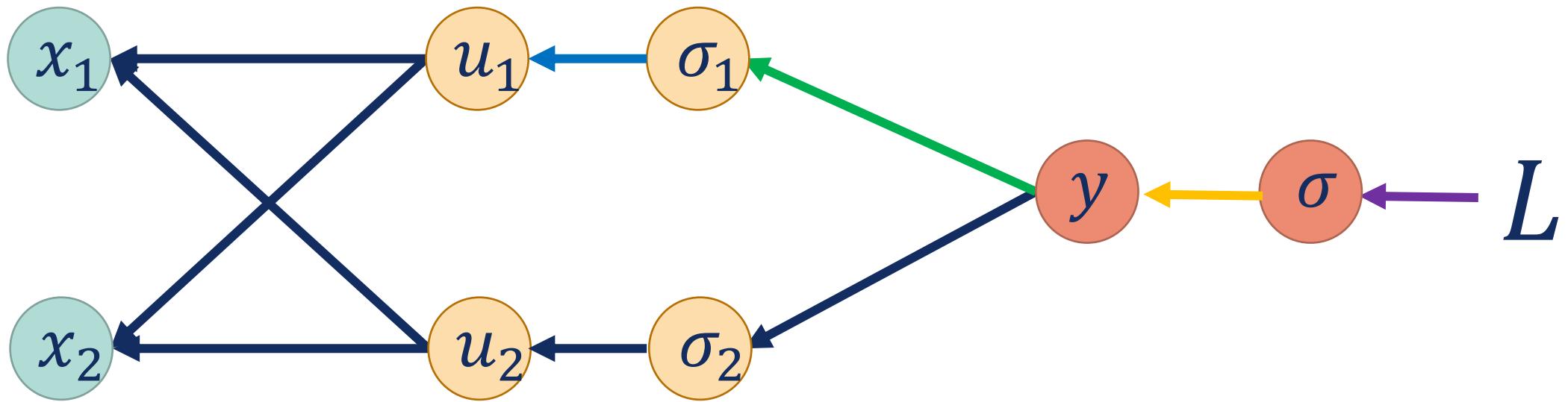


Если каждая из производных небольшая по модулю – произведение тоже будет маленьким. Чем больше слоев, тем меньше.

Значит в «глубине» веса не будут меняться!

Взрыв градиентов (Gradient explosion)

Аналогично для больших по модулю производных:



Модуль произведения производных растет экспоненциально с числом слоев. Чем больше слоев, тем больше будут градиенты.

В «глубине» веса будут кидать из стороны в сторону с огромным шагом.

Deep learning: что изменилось?

Нейросети и backpropagation известны давно, почему же последние достижения происходят только сейчас?

Два фактора:

1. Выросли вычислительные мощности и развились вычисления на GPU
2. Тем временем люди придумали много полезных трюков и архитектур

Архитектуры нейросетей

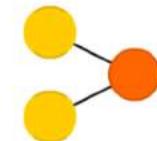
<http://www.asimovinstitute.org/neural-network-zoo/>

- Backfed Input Cell
- Input Cell
- △ Noisy Input Cell
- Hidden Cell
- Probabilistic Hidden Cell
- △ Spiking Hidden Cell
- Output Cell
- Match Input Output Cell
- Recurrent Cell
- Memory Cell
- △ Different Memory Cell

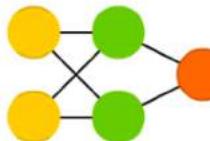
A mostly complete chart of Neural Networks

©2016 Fjodor van Veen - asimovinstitute.org

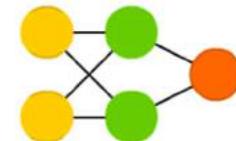
Perceptron (P)



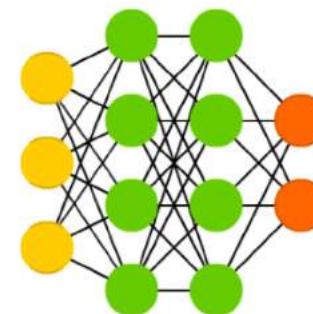
Feed Forward (FF)



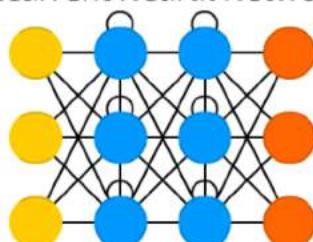
Radial Basis Network (RBF)



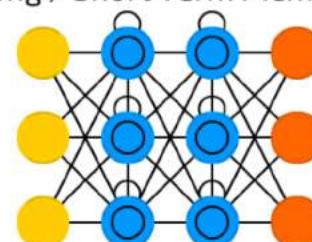
Deep Feed Forward (DFF)



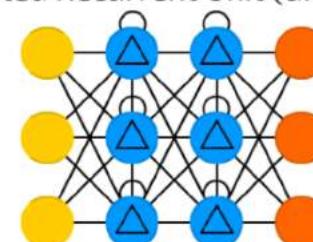
Recurrent Neural Network (RNN)



Long / Short Term Memory (LSTM)



Gated Recurrent Unit (GRU)



Auto Encoder (AE)



Variational AE (VAE)



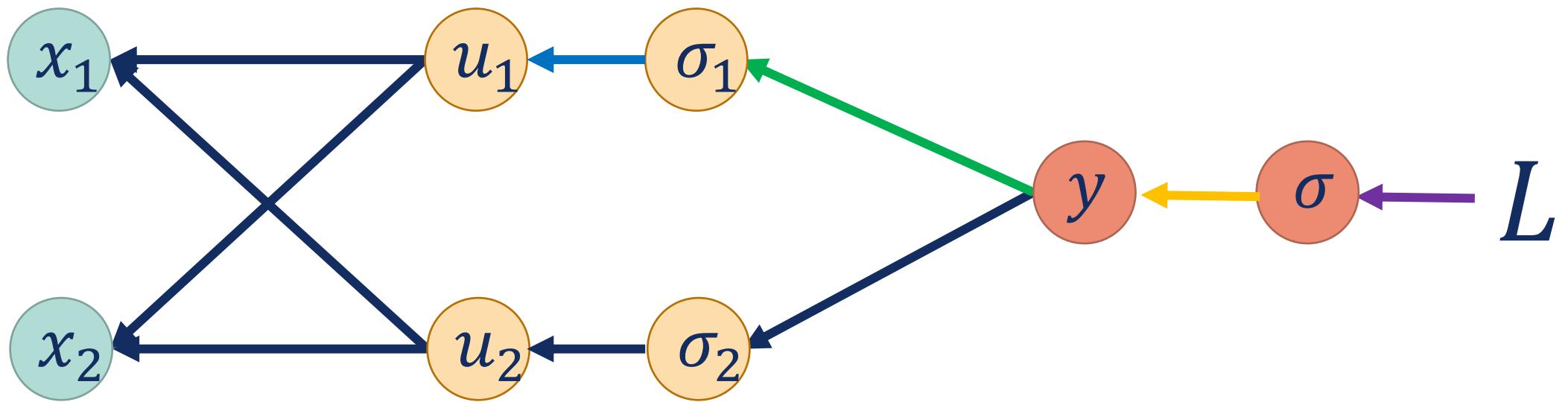
Denoising AE (DAE)



Sparse AE (SAE)

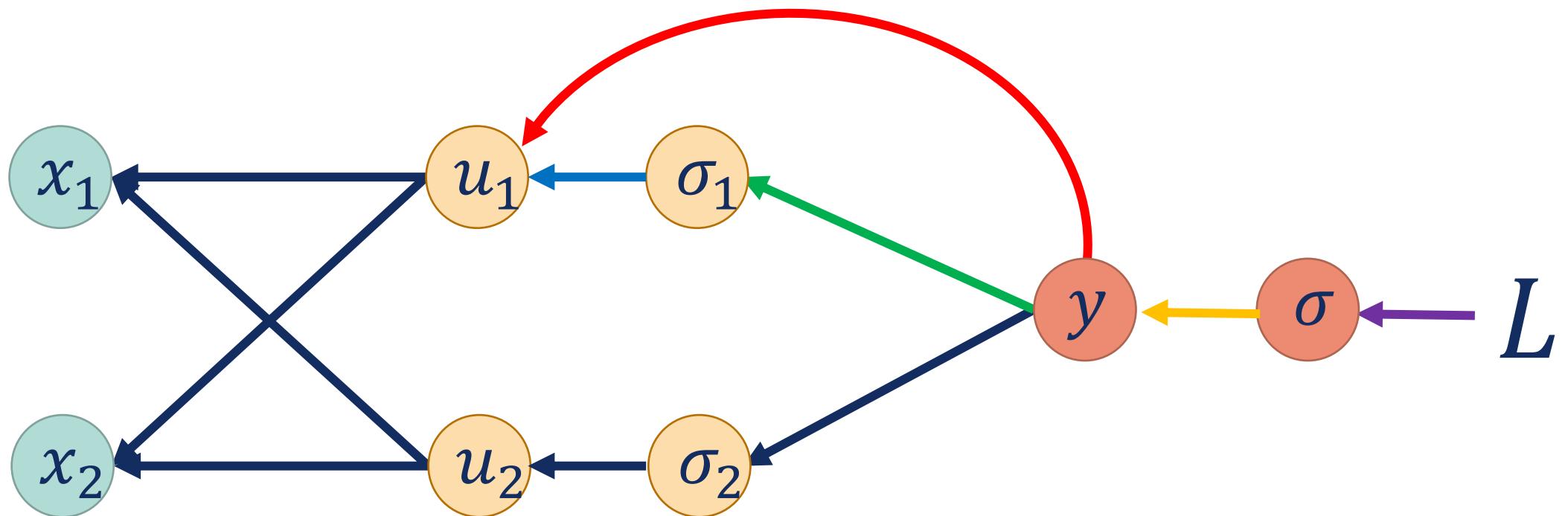


Пример: как архитектура помогает обучению



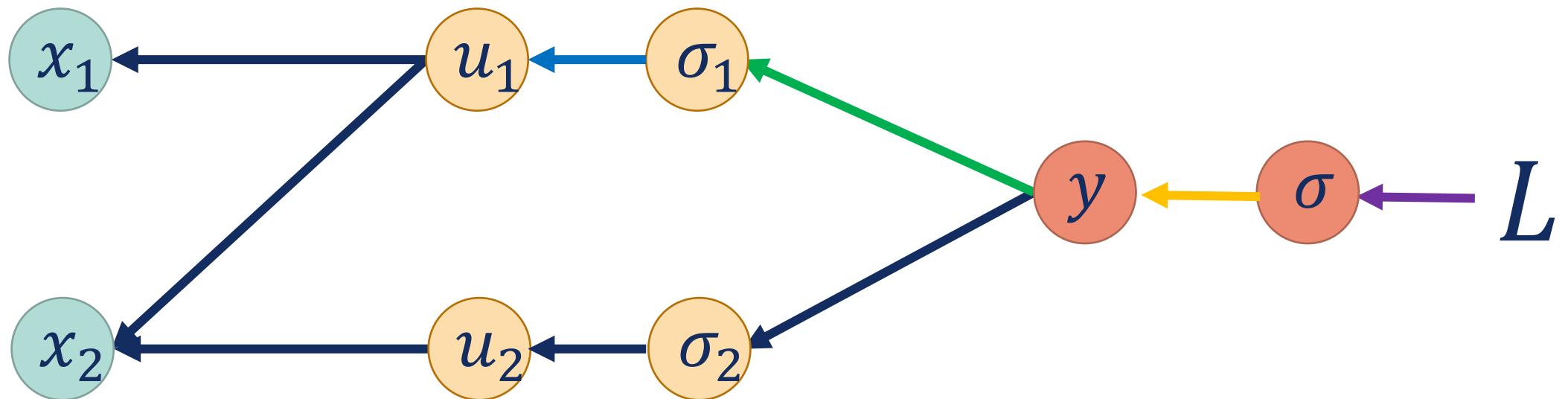
Пример: как архитектура помогает обучению

Добавляем «перепрыгивающие» через слои связи (*residual connection*):
теперь градиент будет «дотекать» не затухая, а большие градиенты
просто будем обрезать (*gradient clipping*)



Пример: как архитектура помогает обучению

Другой вариант: сразу убираем часть связей (считаем, что вес на них ноль и не настраиваем его) – меньше параметров, меньше риск переобучиться



Дальнейшие планы

Далее мы увидим примеры применения этих идей в реально используемых архитектурах нейросетей:

- в оставшейся части лекции мы рассмотрим свёрточные нейросети,
- в следующий раз – закончим обсуждать свёрточные сети, разберем рекуррентные и, обзорно, другие интересные примеры архитектур

3. Свёрточные нейросети

Сверточные нейросети: задачи

- Хорошо проявили себя в компьютерном зрении
- Сейчас применяются также в распознавании речи и анализе текстов на естественном языке
- Мы начнем знакомство с приложений в компьютерном зрении

Частые задачи в Computer Vision

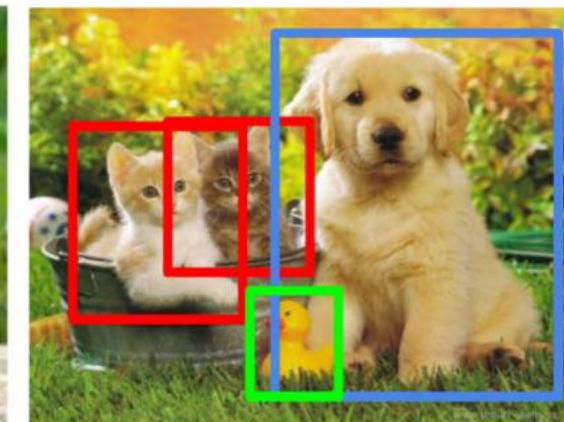
Classification



Classification + Localization



Object Detection



Instance Segmentation



CAT

CAT

CAT, DOG, DUCK

CAT, DOG, DUCK

Single object

Multiple objects

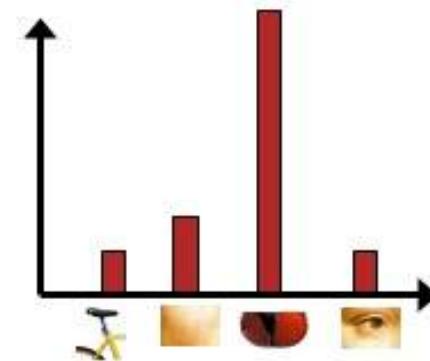
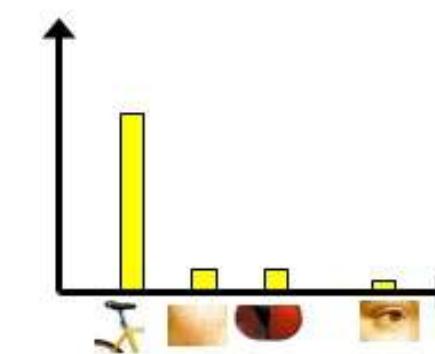
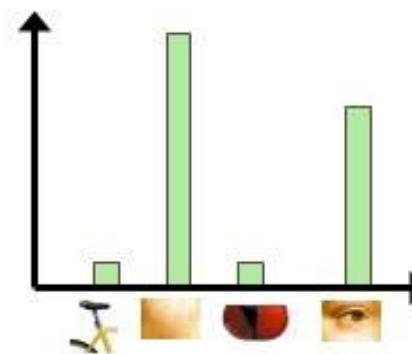
На чем остановимся сначала

Classification

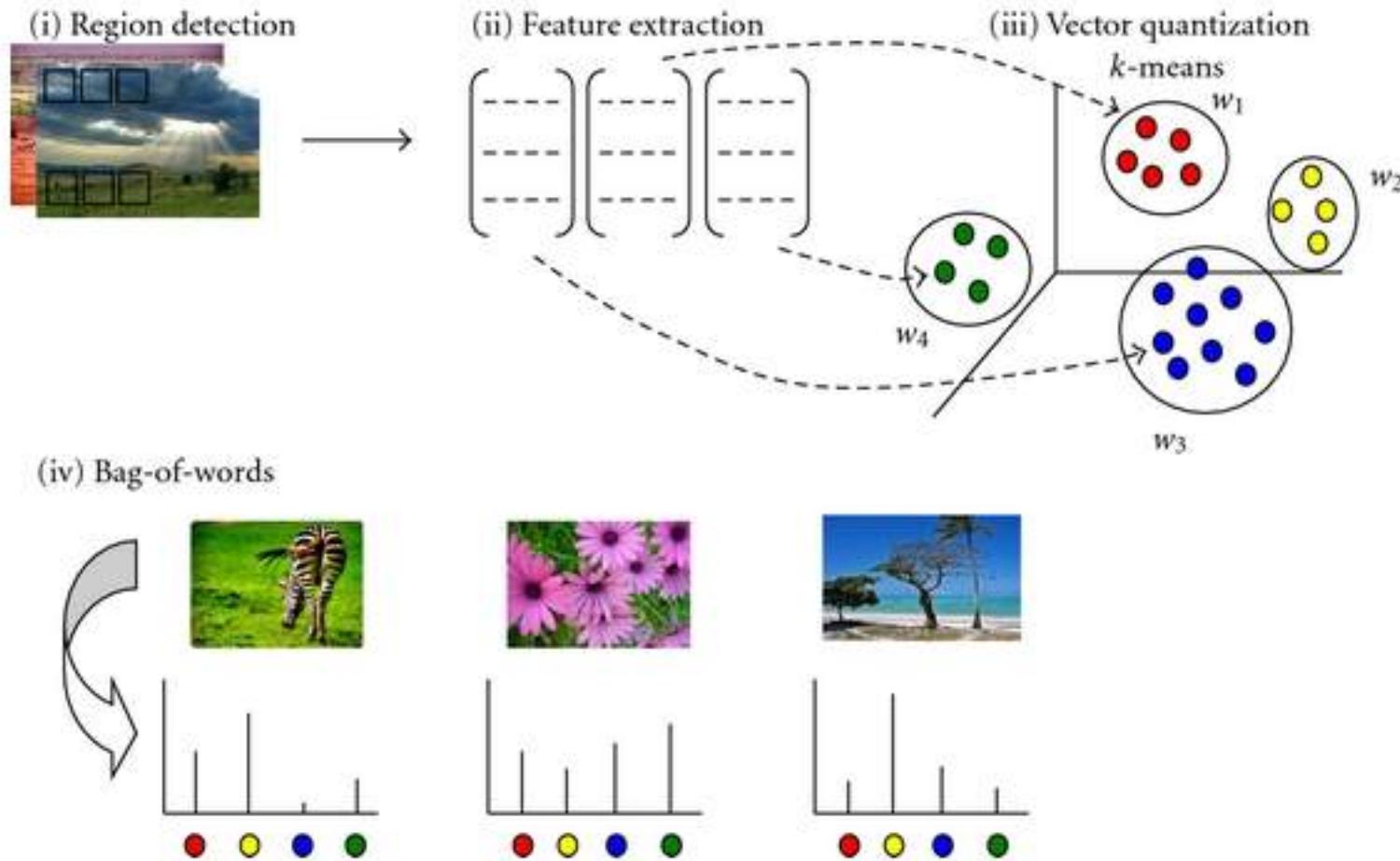


CAT

С этой задачей мы уже сталкивались



Вспоминаем bag of visual words

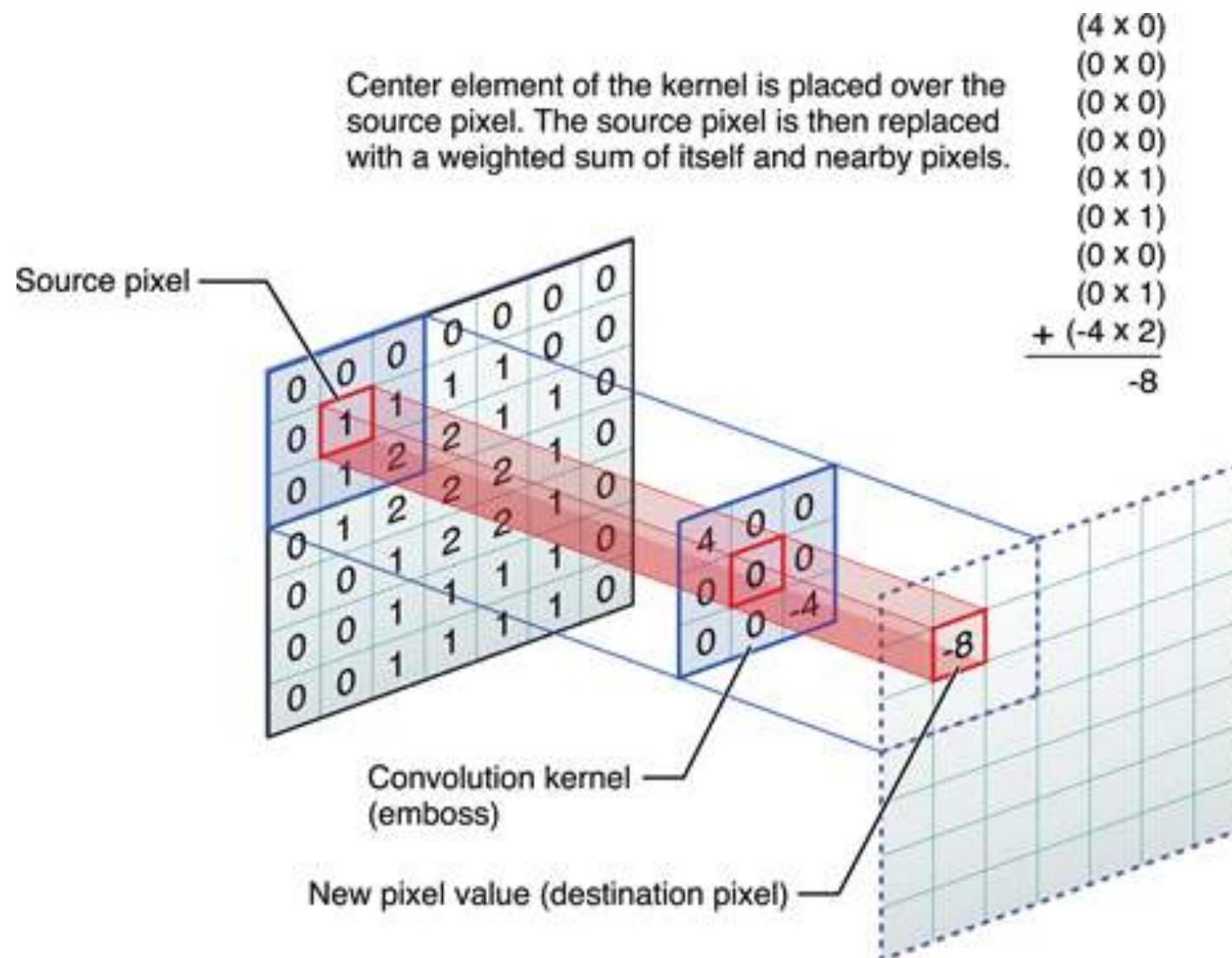


Как использовать идею в нейросетях?

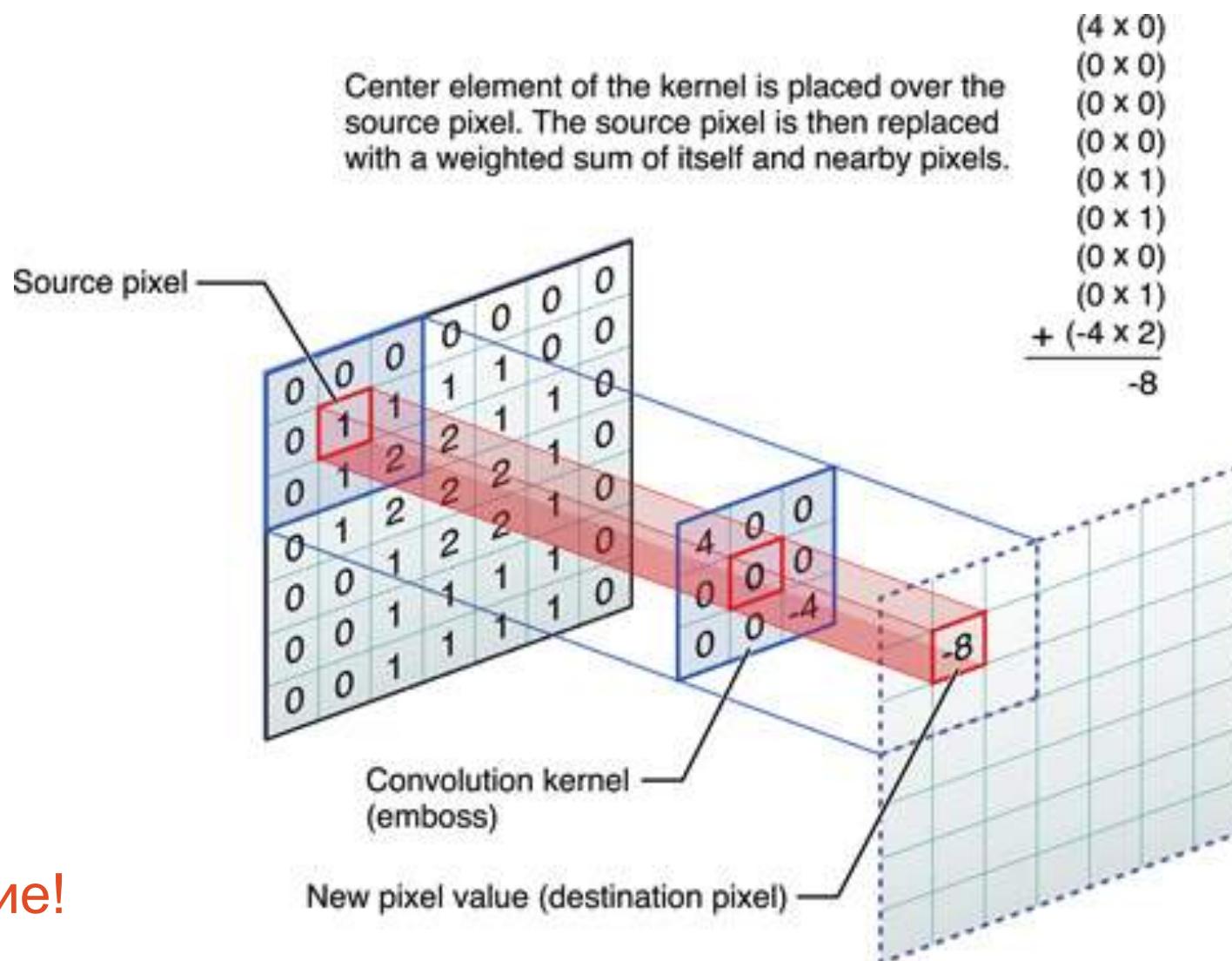
- Классифицируемое изображение – это матрица с числами (яркостями пикселей, если ч/б формат)
- Паттерны (фильтры), которые помогут понять, что именно на изображении – тоже матрицы с яркостями пикселей
- Что будет, если приложить паттерн к фрагменту изображения, перемножить пиксели паттерна и изображения и сложить произведения?



Как применяем фильтр к изображению



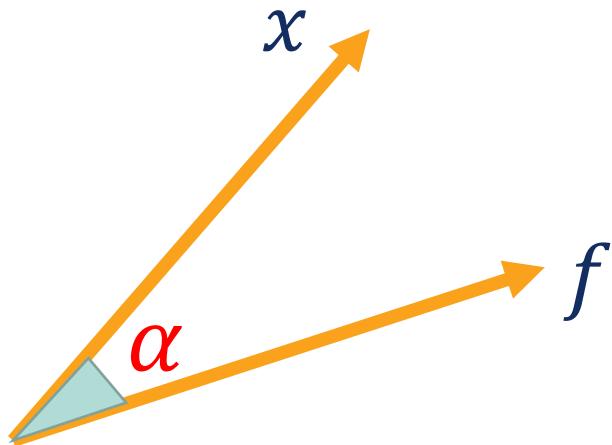
Операция свертки с фильтром



Это же
скалярное
произведение!

Напоминание: скалярное произведение

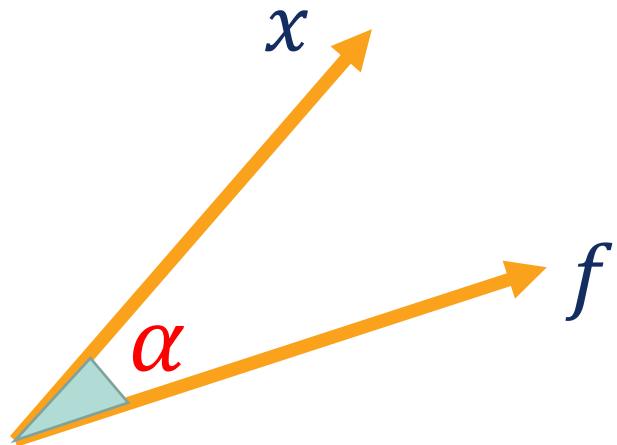
Пусть оба вектора отнормированы (имеют единичную длину).
Когда их скалярное произведение максимально?



Напоминание: скалярное произведение

Пусть оба вектора отнормированы (имеют единичную длину).

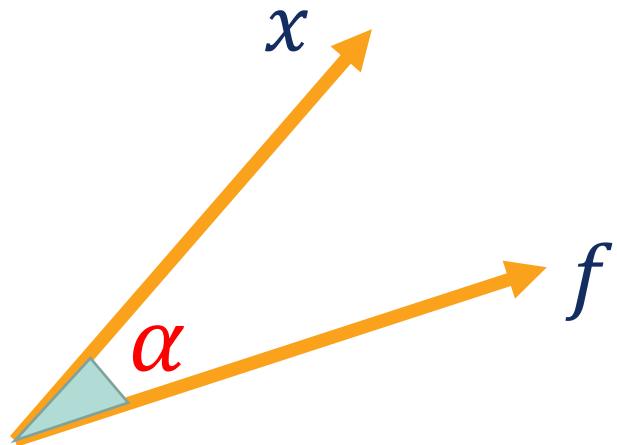
Когда их скалярное произведение максимально?



$$\begin{aligned}\langle f, x \rangle &= \|f\| \|x\| \cos \alpha = \cos \alpha \\ \cos \alpha &= 1 \\ \alpha &= 0\end{aligned}$$

Напоминание: скалярное произведение

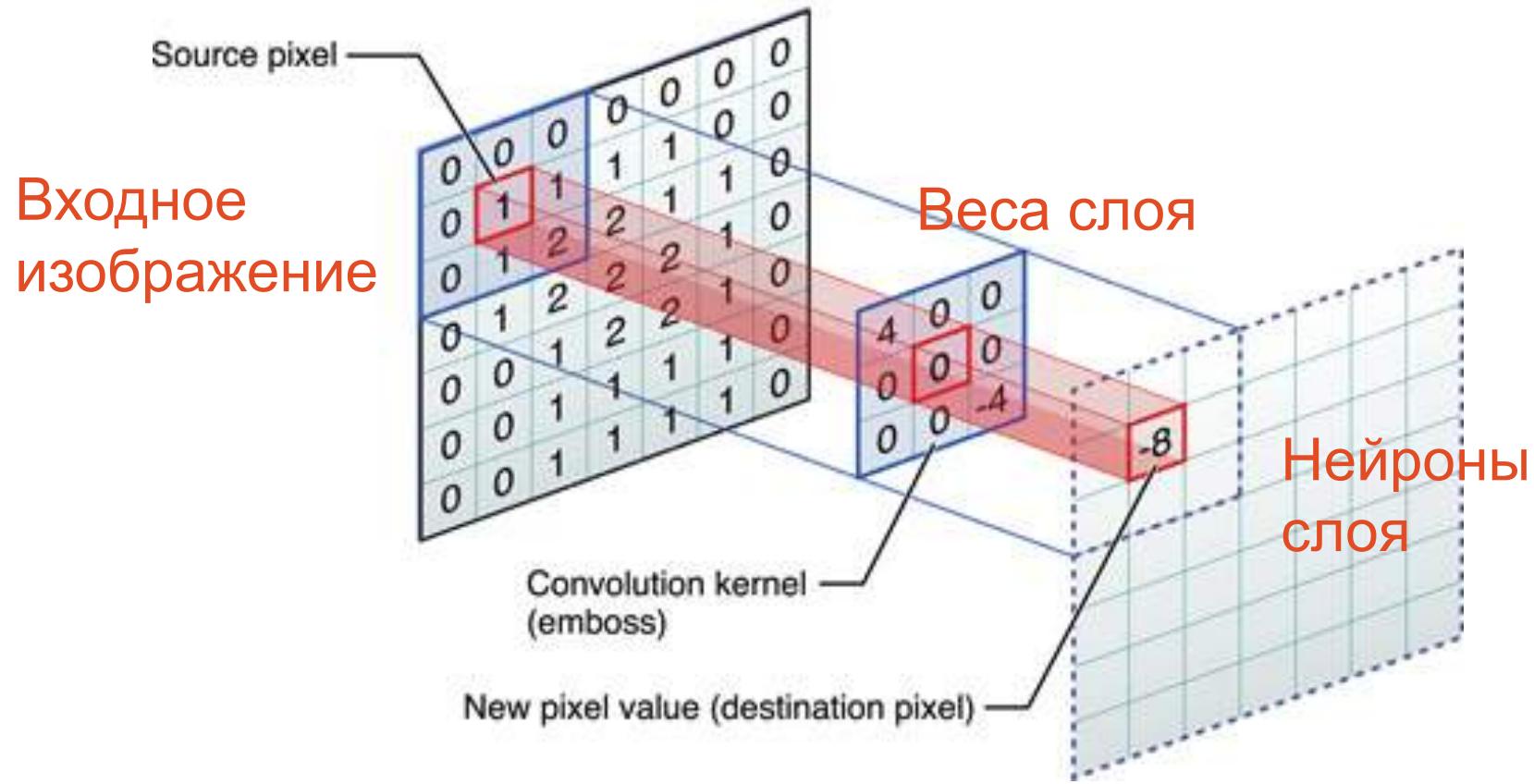
Пусть оба вектора отнормированы (имеют единичную длину).
Когда их скалярное произведение максимально?



$$\begin{aligned}\langle f, x \rangle &= \|f\| \|x\| \cos \alpha = \cos \alpha \\ \cos \alpha &= 1 \\ \alpha &= 0\end{aligned}$$

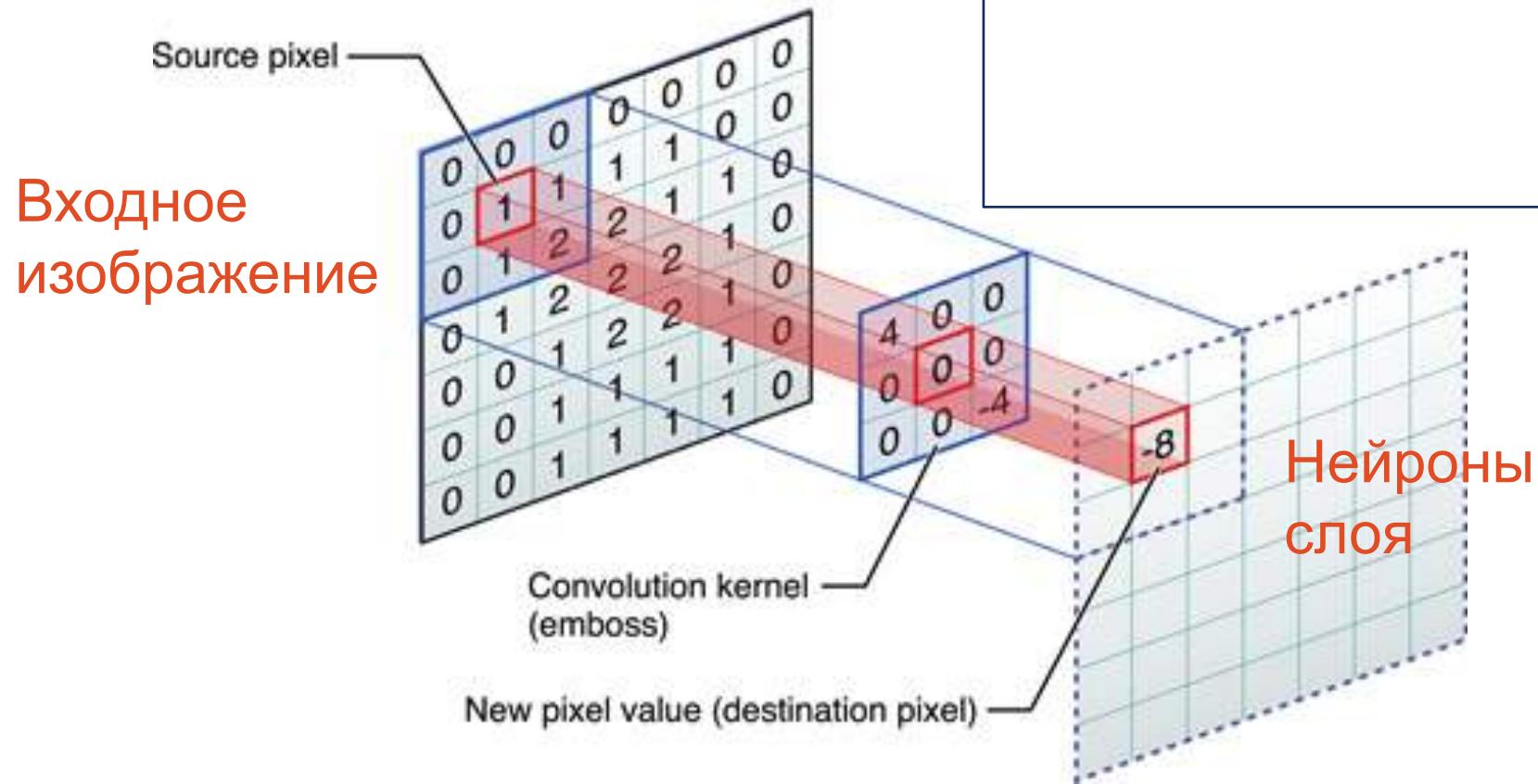
Поэтому свертка с фильтром просто вычисляет похожесть фрагмента изображения на фильтр

Идея свёрточного слоя



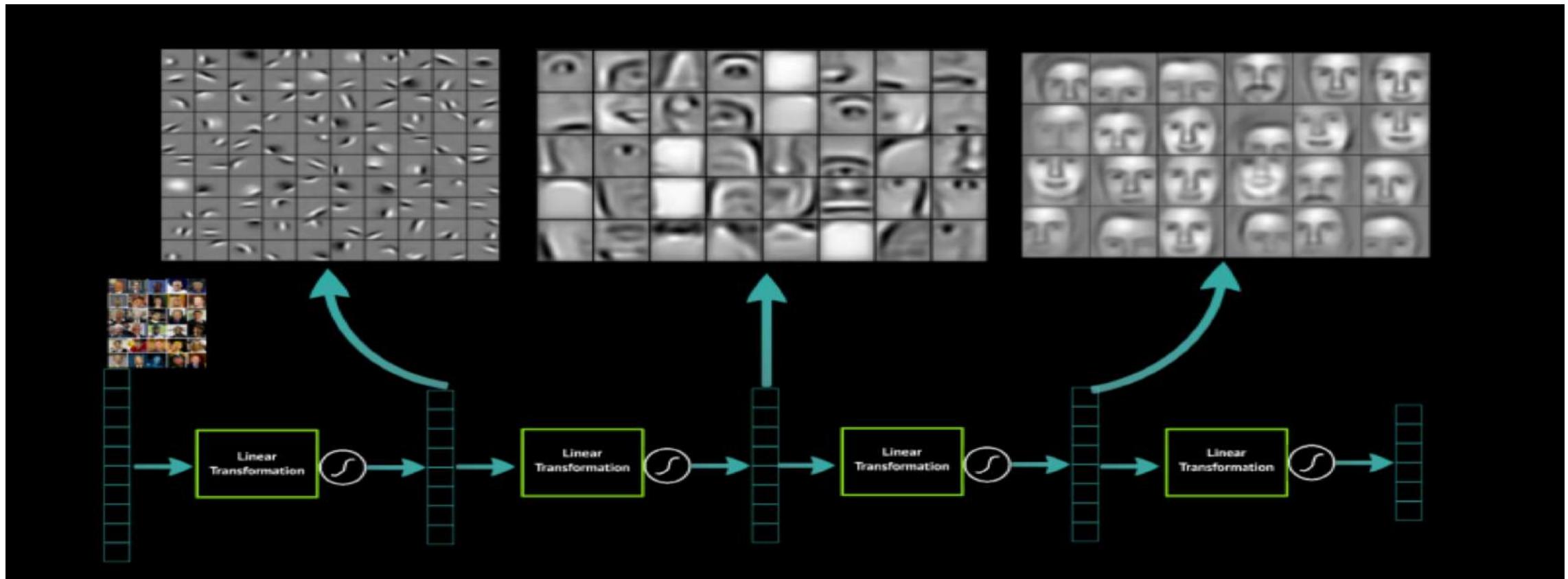
Идея свёрточного слоя

1. Т.к. фильтр – это настраиваемые backprop веса, то сеть сама «подберет» визуальные слова

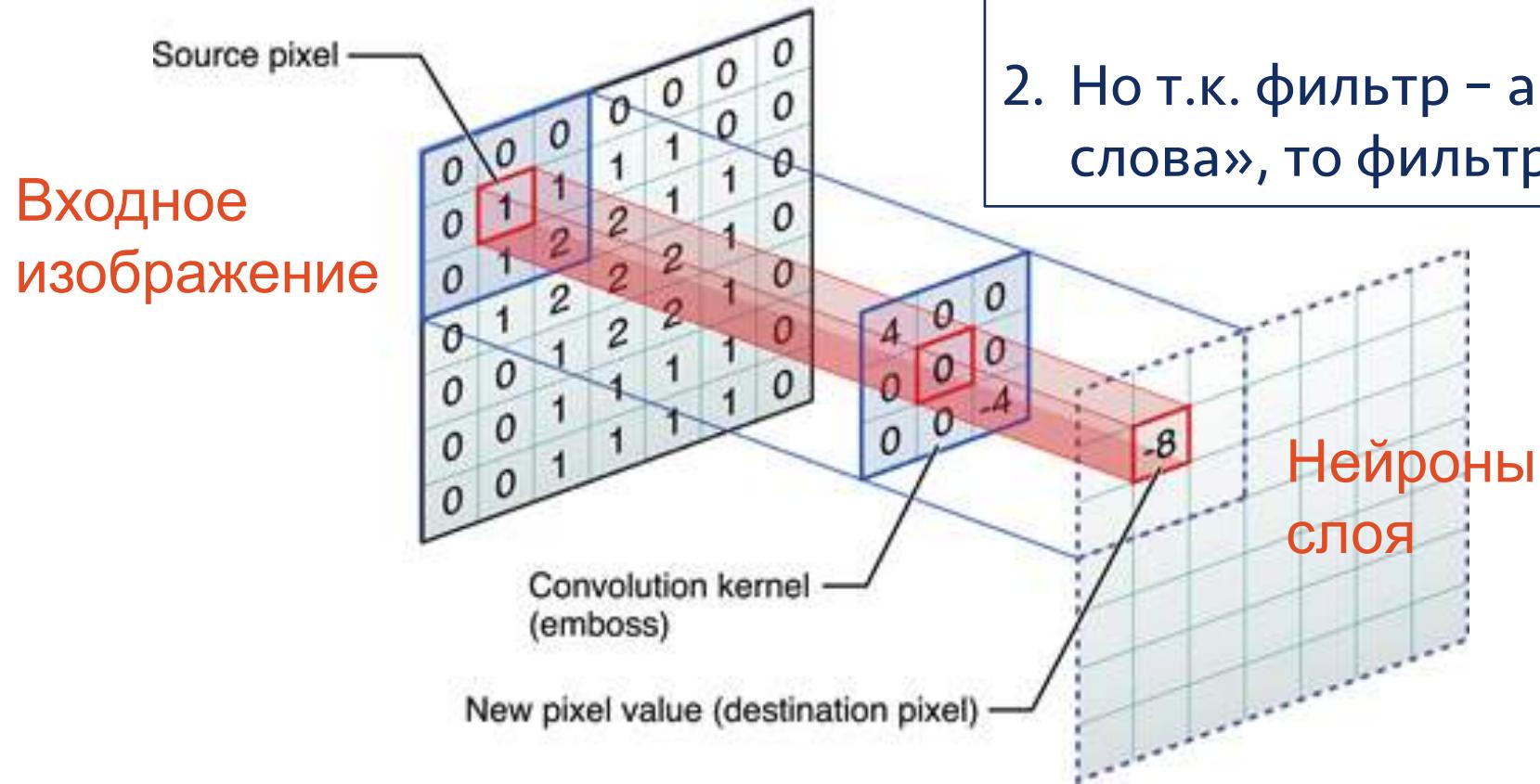


Пример: распознавание лиц

Backprop действительно неплохо подбирает фильтры:



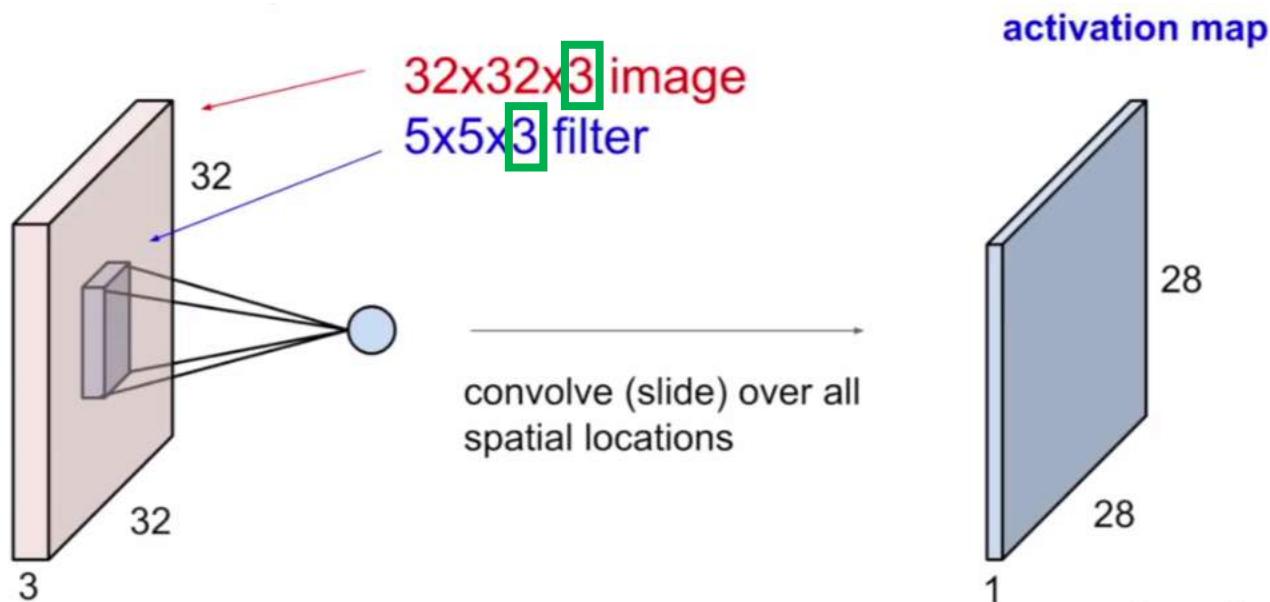
Идея свёрточного слоя



1. Т.к. фильтр – это настраиваемые backprop веса, то сеть сама «подберет» визуальные слова
2. Но т.к. фильтр – аналог «визуального слова», то фильтров нужно **больше**

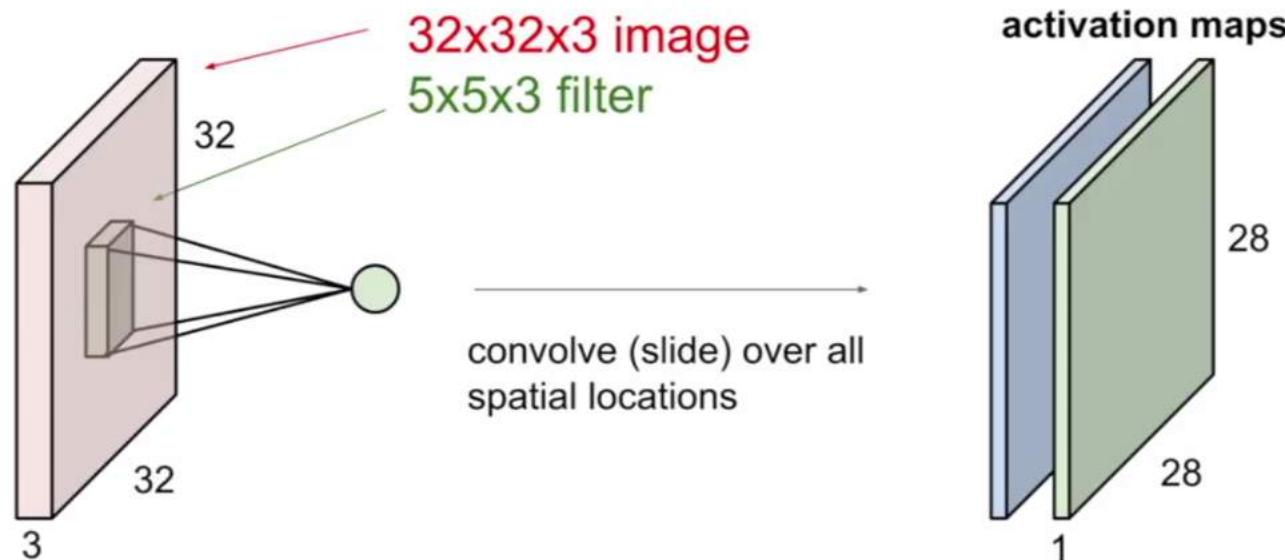
Толщина сверток и их количество

1. Свертка не плоская, а такой же «толщины» как исходное изображение



Толщина сверток и их количество

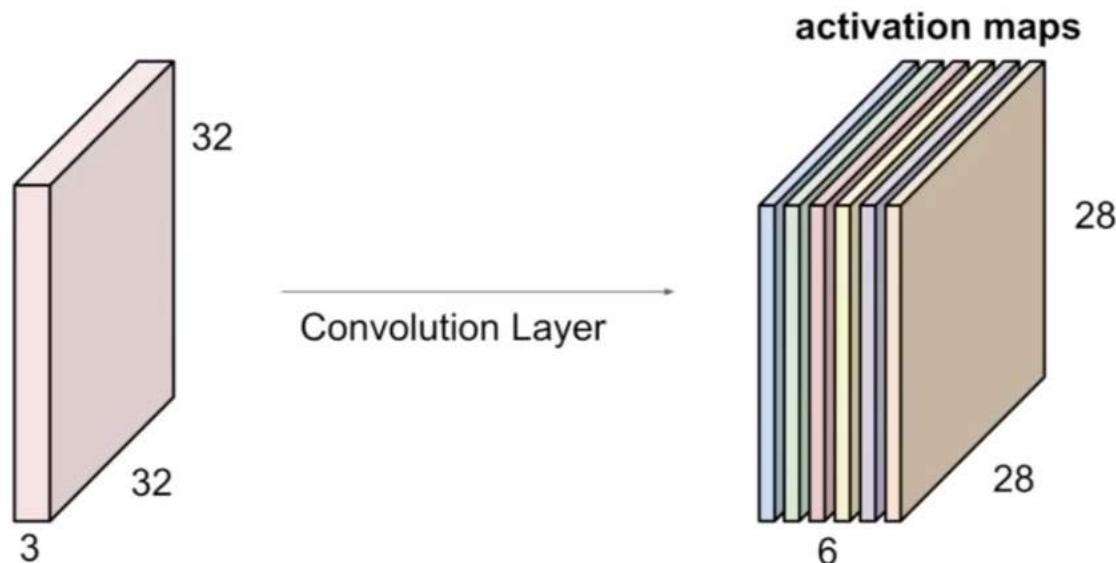
1. Свертка не плоская, а такой же «толщины» как исходное изображение



2. Каждая свертка порождает еще одну карту активации (карту признаков)

Толщина сверток и их количество

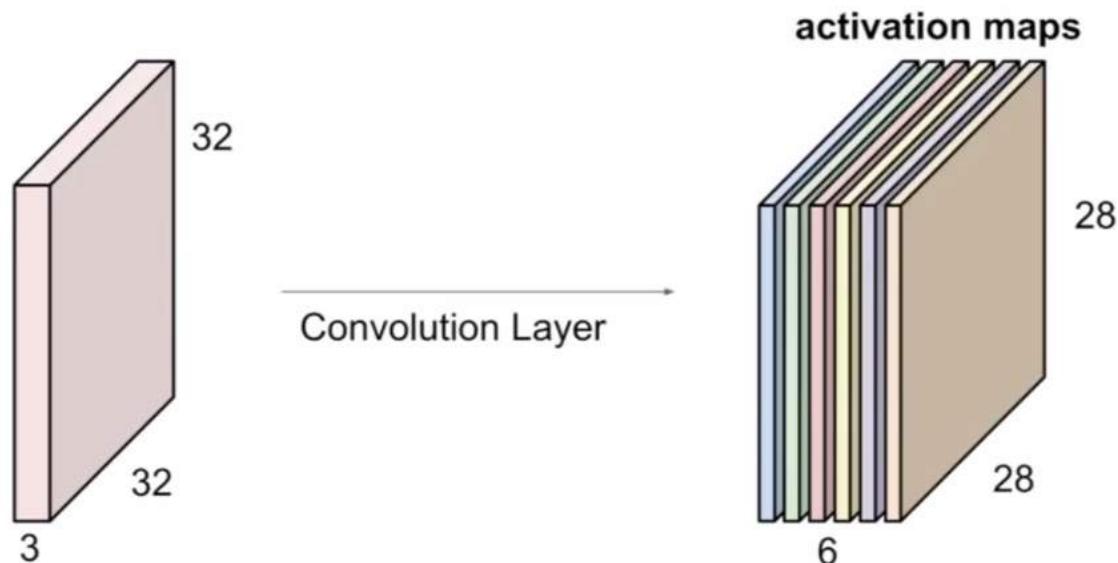
1. Свертка не плоская, а такой же «толщины» как исходное изображение



2. Каждая свертка порождает еще одну карту активации (карту признаков)
3. В одном слое делают много фильтров (а, значит, много карт активации)

Толщина сверток и их количество

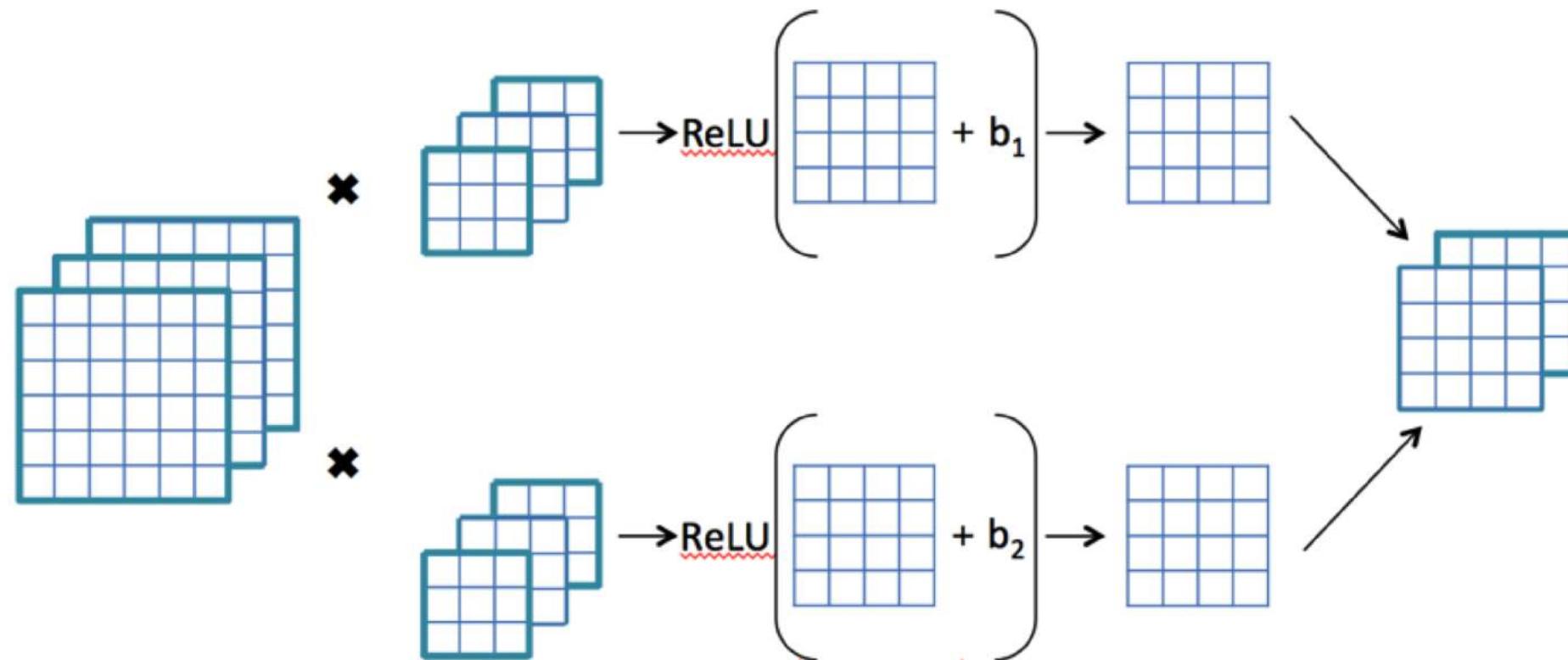
1. Свертка не плоская, а такой же «толщины» как исходное изображение



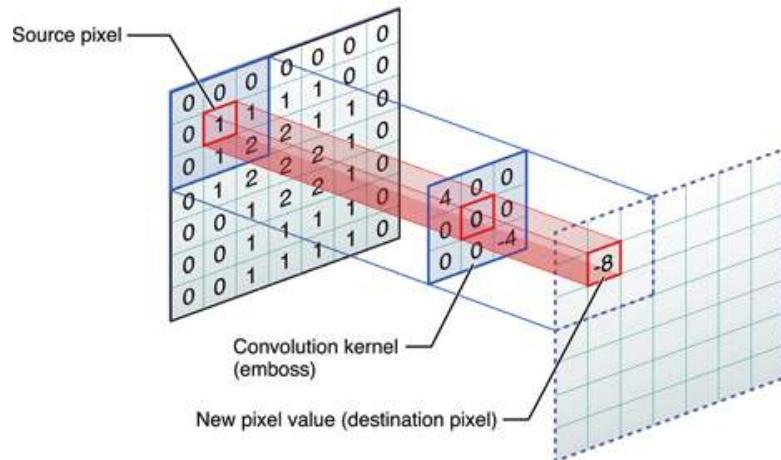
2. Каждая свертка порождает еще одну карту активации (карту признаков)
3. В одном слое делают много фильтров (а, значит, много карт активации)
4. Можно считать, что теперь в изображении столько каналов (такая толщина) – значит такой будет толщина фильтров в следующем слое

Сдвиги и нелинейности

К результатам свертки также, как это было в полносвязных сетях, добавляется сдвиг (порог) b и результат также подается на вход нелинейности (например ReLU)

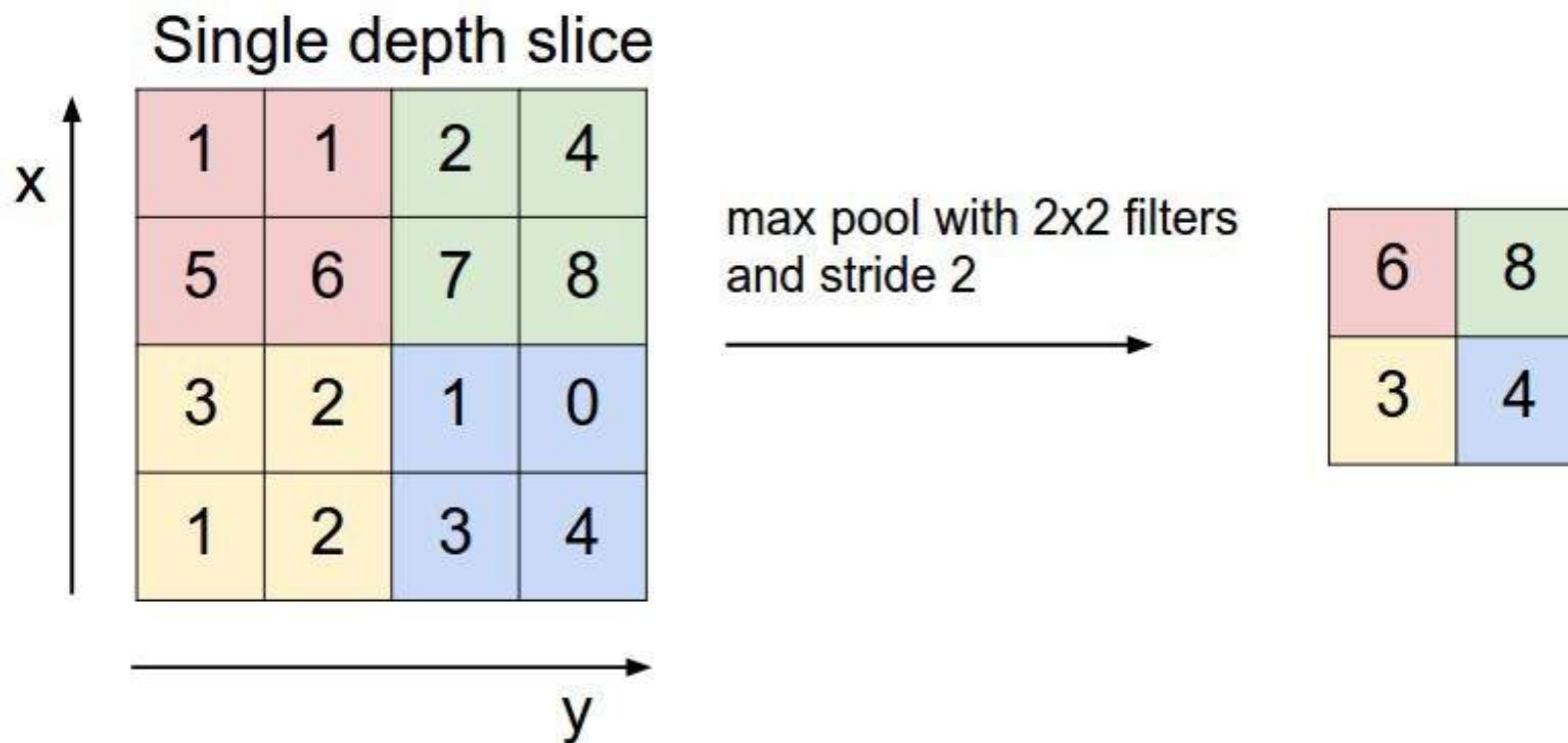


Нужно ли что-то еще кроме сверток?

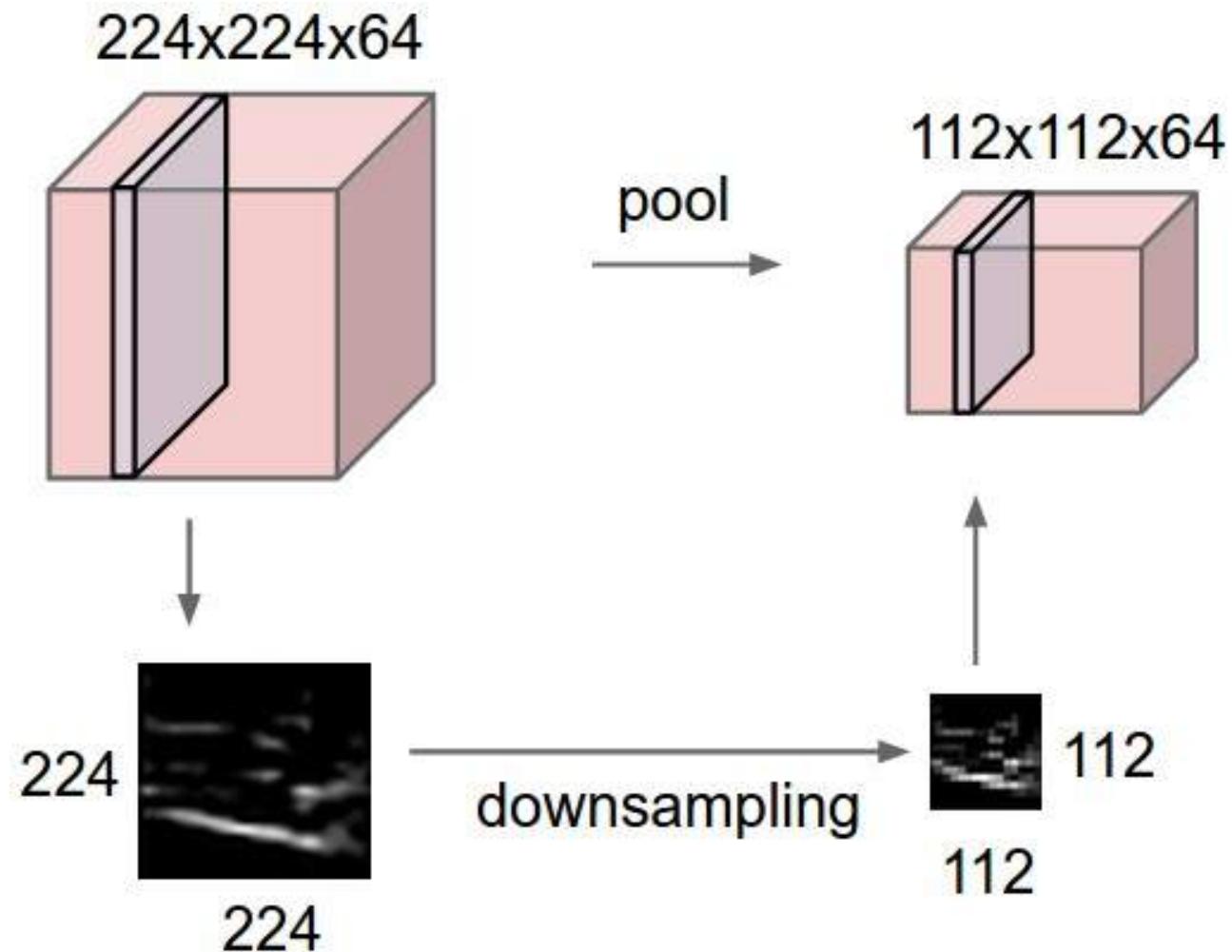


- Размер изображения после свертки такой же или почти такой же
- Размерность не уменьшается, а еще и фильтров будет много – значит только увеличится
- Получается очень много параметров – есть риск не получить никакой обобщающей способности у алгоритма

Слой пулинга

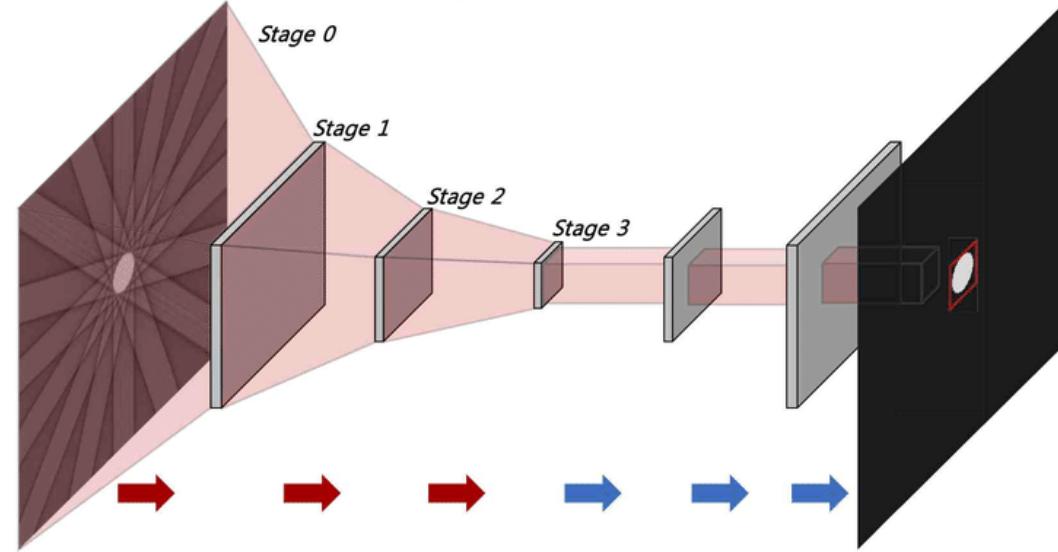


Пуллинг: что происходит с изображением

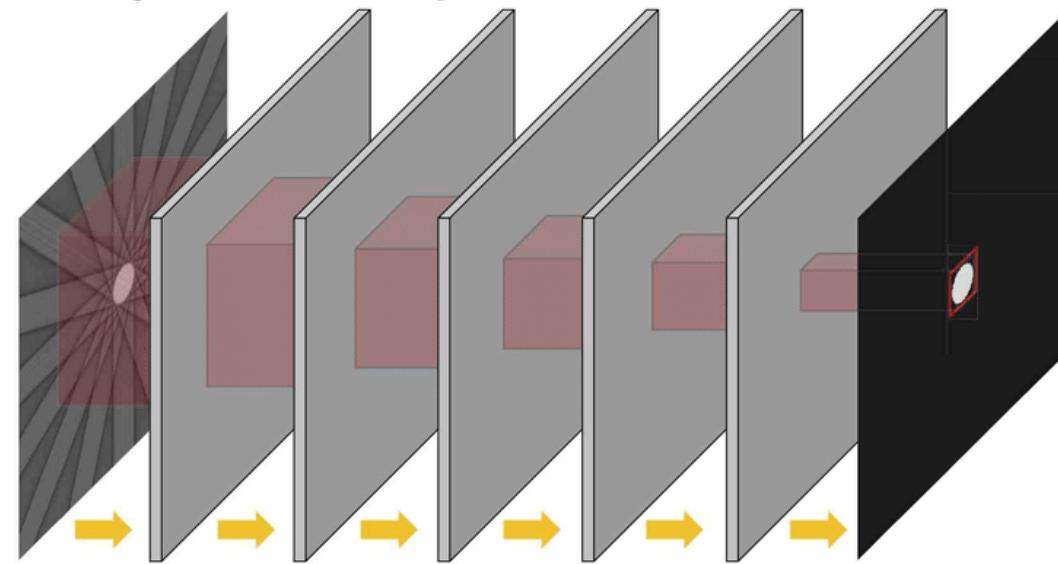


Еще одна
причина
использовать
пулинг

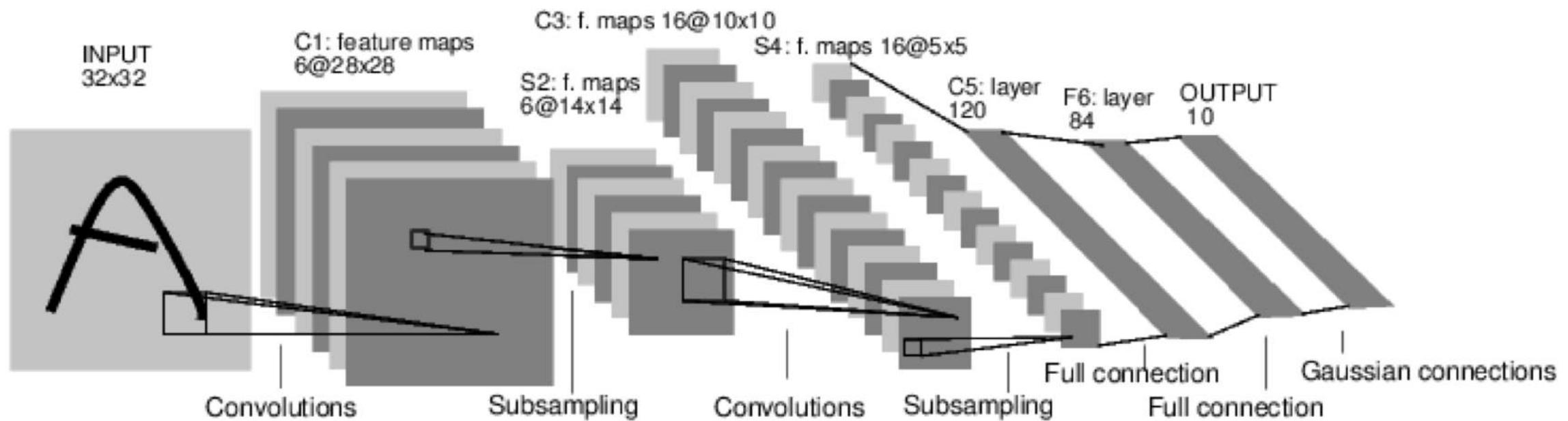
(a) Depth-wise receptive field



(b) Layer-wise receptive field



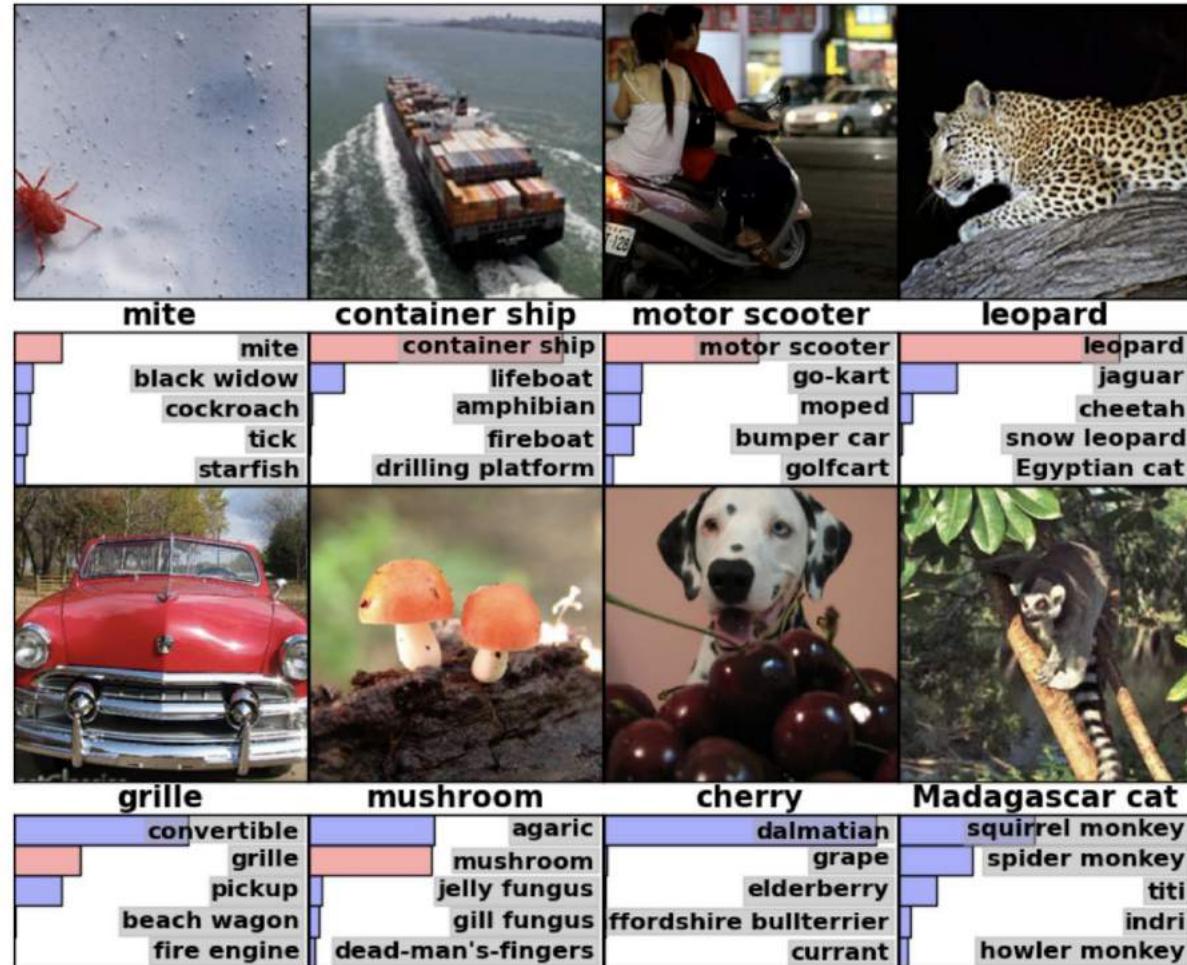
Первые свёрточные сети: LeNet-5



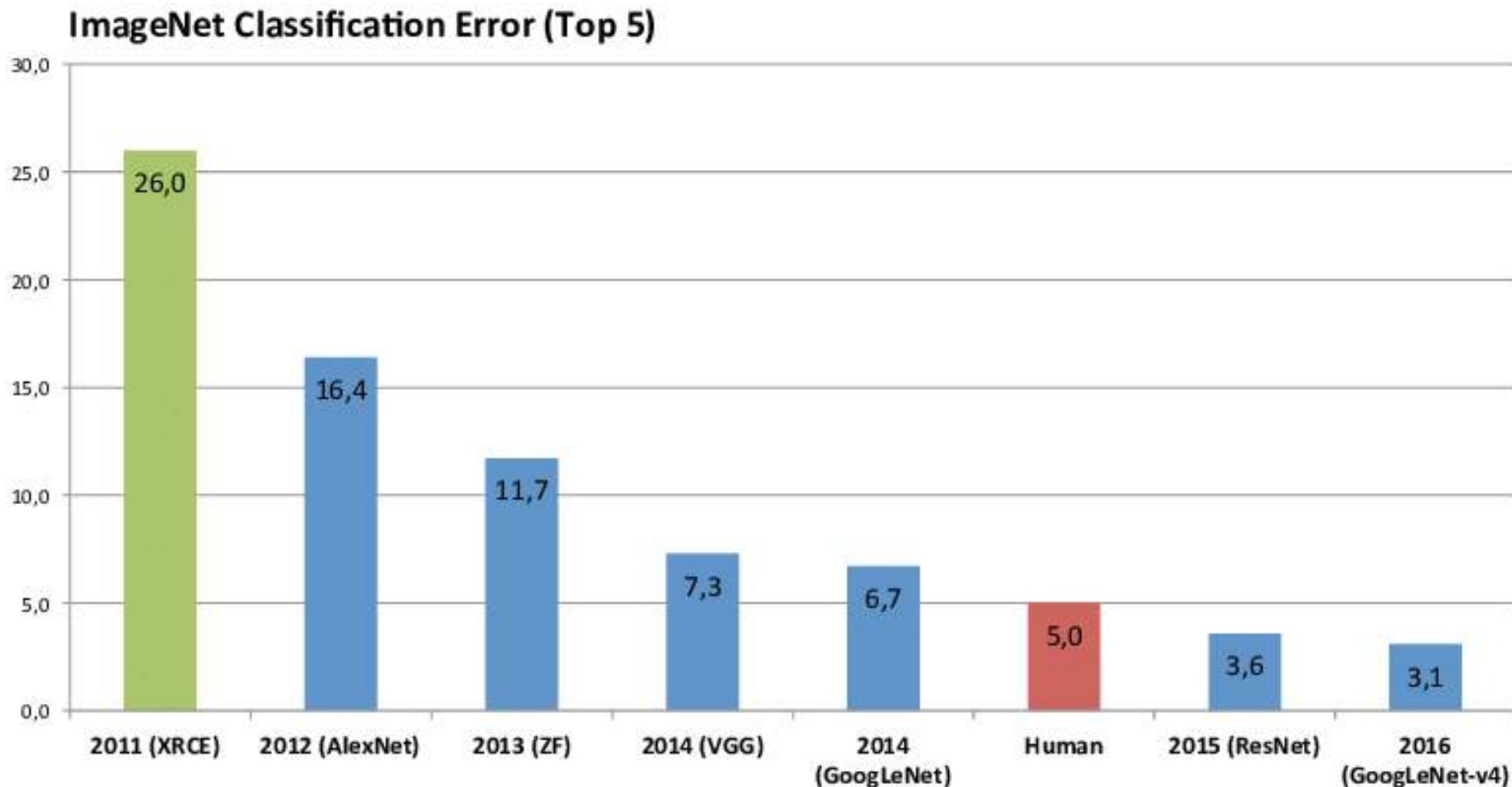
Соревнования по компьютерному зрению

IMAGENET

- 1,000 object classes (categories).
- Images:
 - 1.2 M train
 - 100k test.

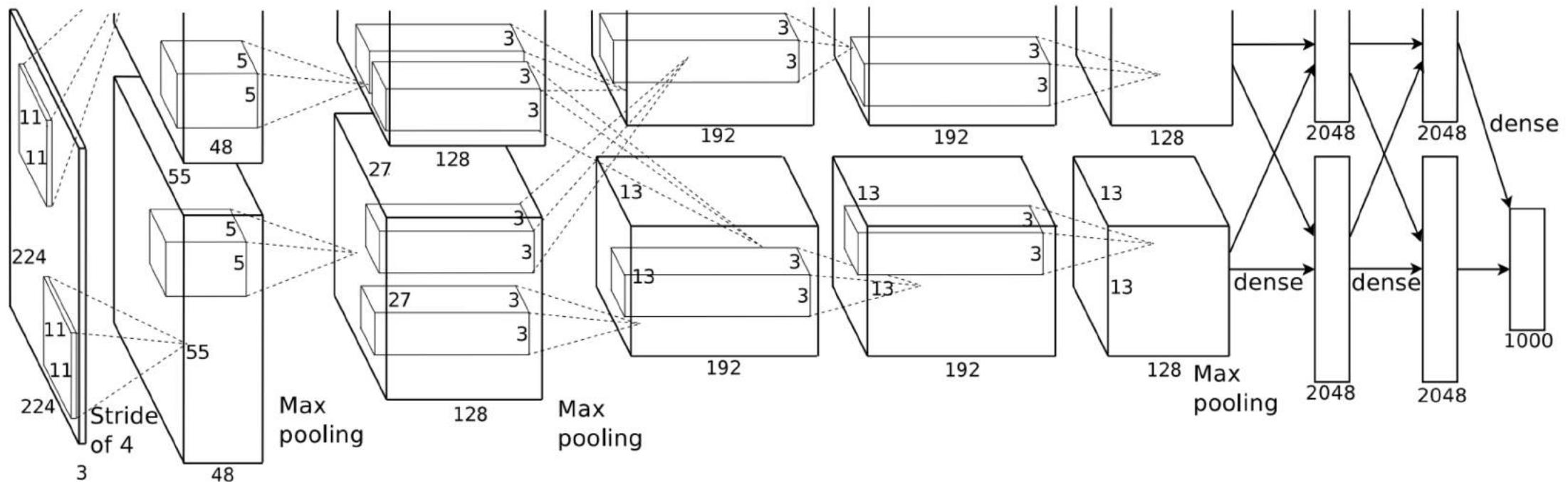


Архитектуры по годам

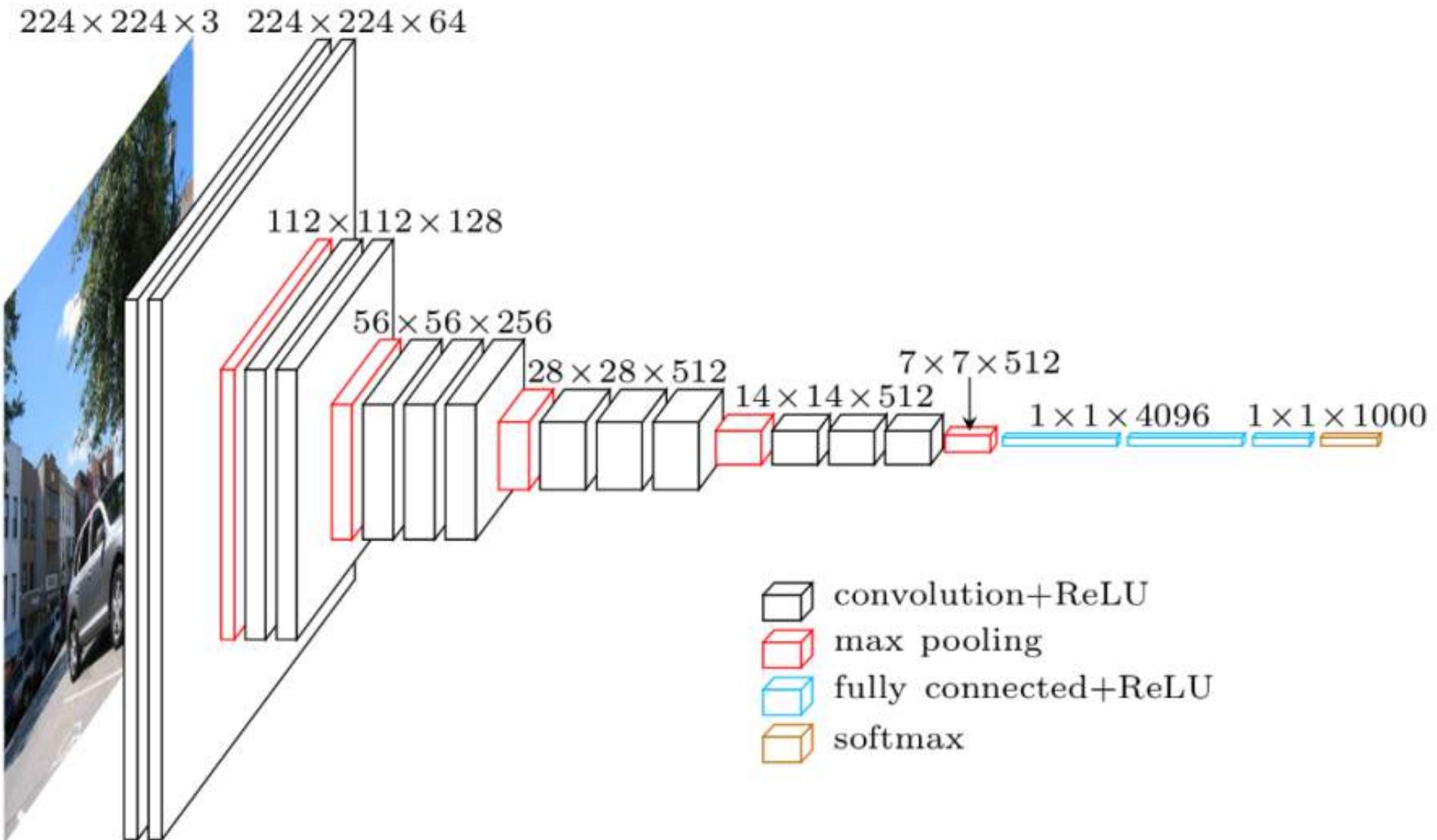


Первые победы: AlexNet

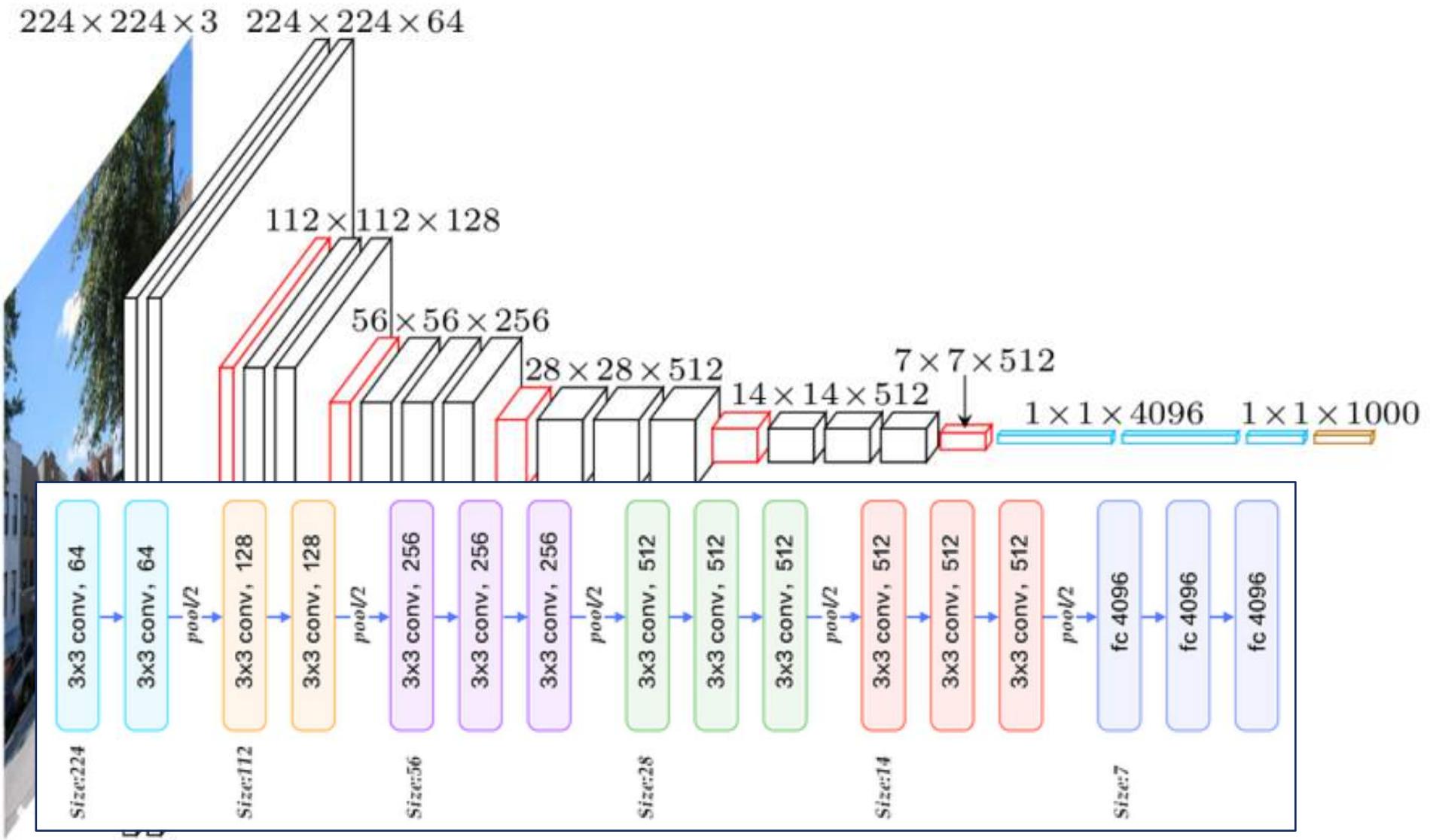
В оригинальной статье верхняя часть картинки была обрезана, в итоге все теперь вставляют в презентации схему в таком виде :)



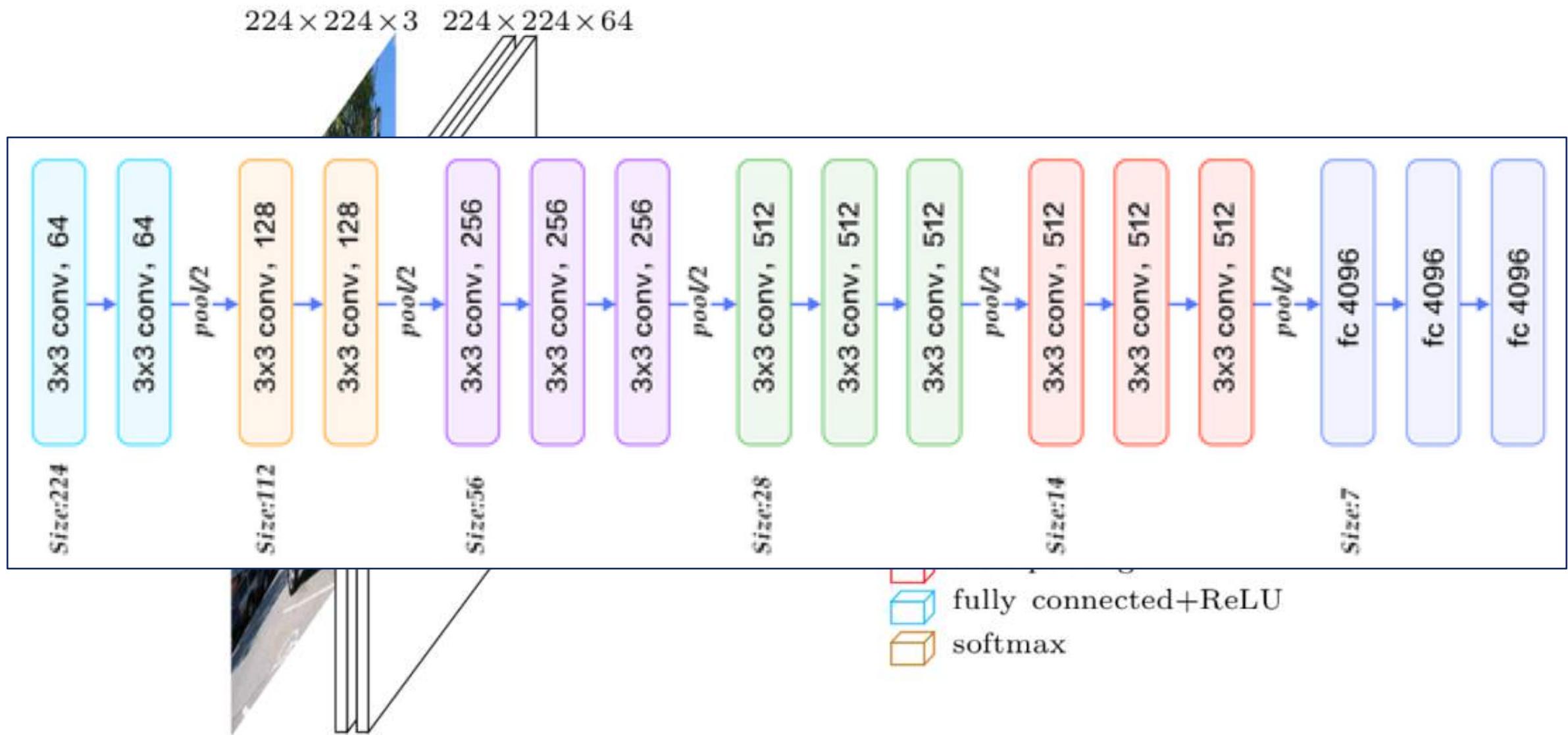
VGG



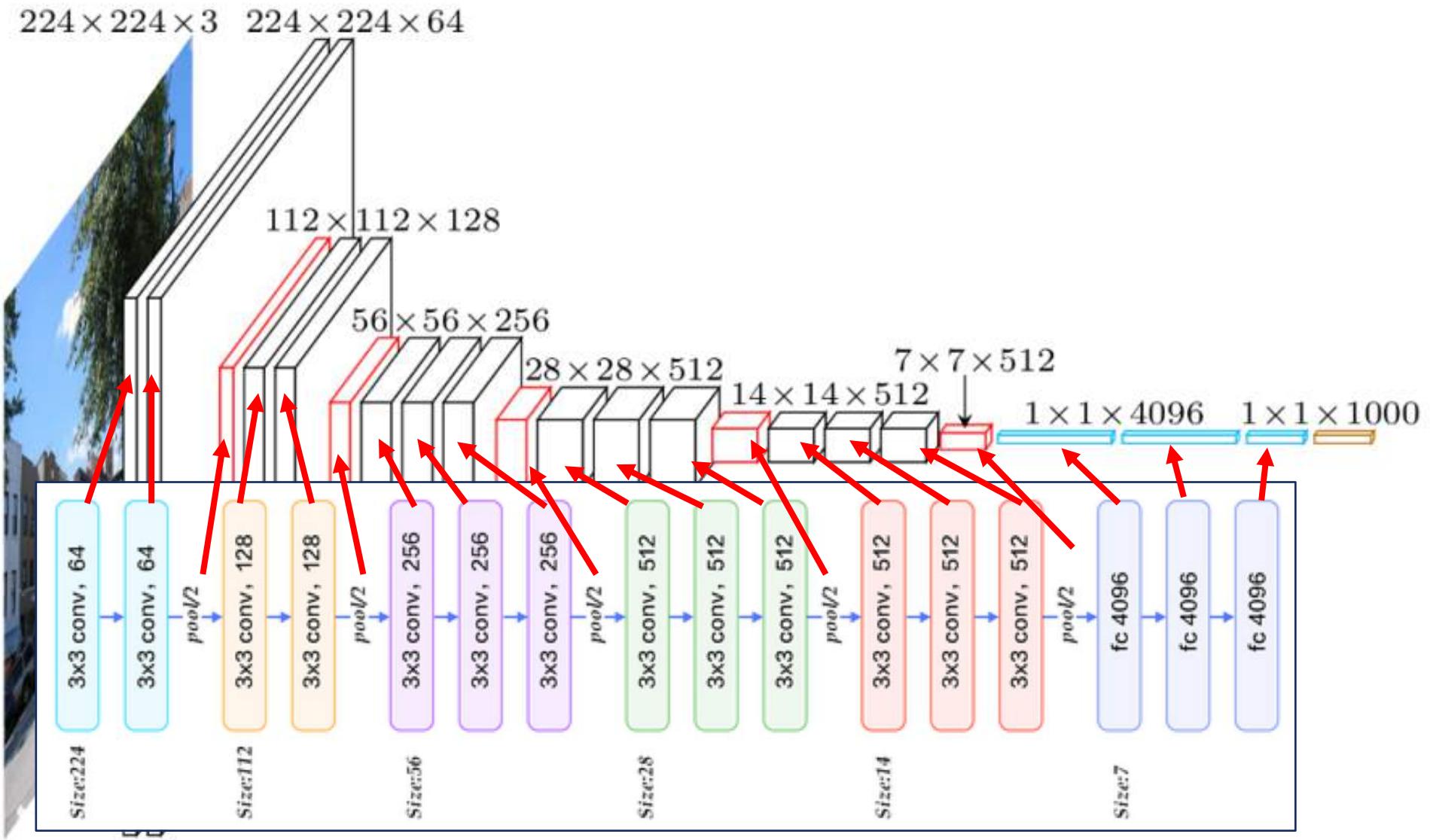
VGG



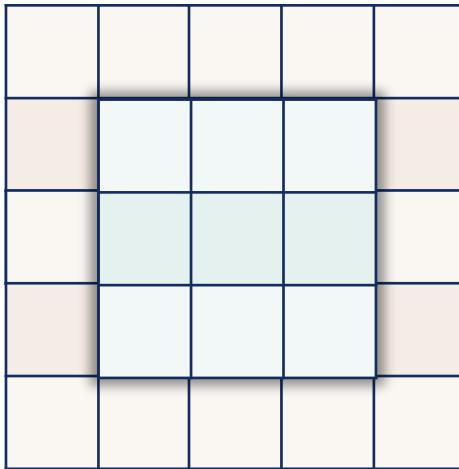
VGG



VGG

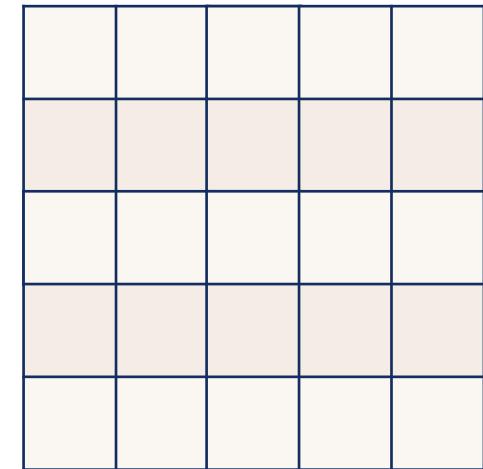


Почему используют фильтры 3x3



Два слоя свертки 3x3
(друг за другом)

$$9 + 1 + 9 + 1 = 20 \text{ параметров}$$

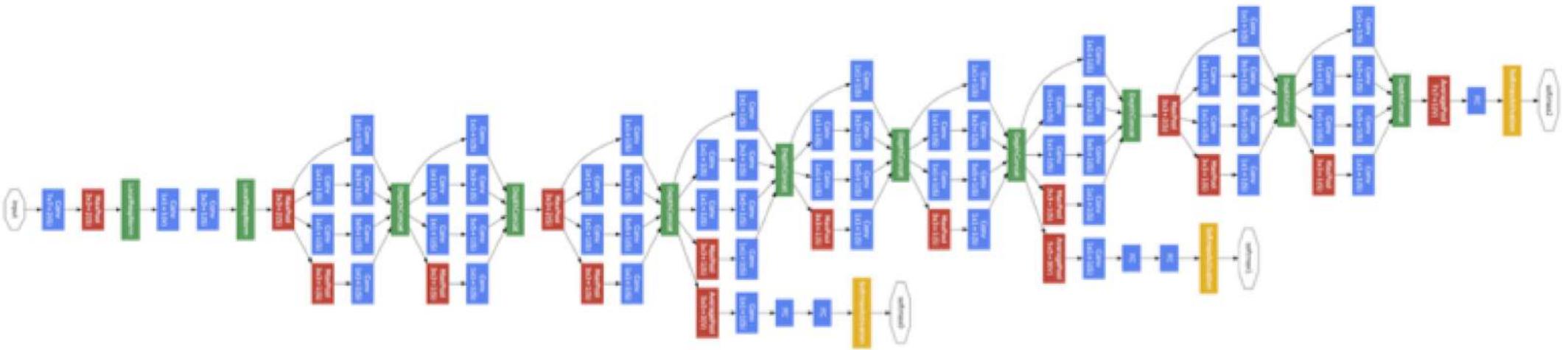


Один слой свертки 5x5

$$25 + 1 = 26 \text{ параметров}$$

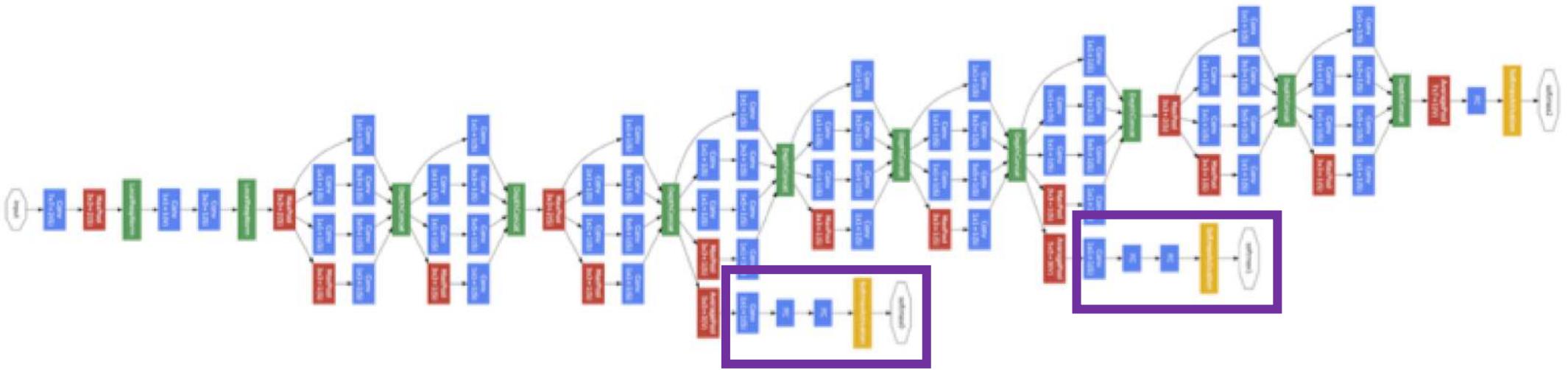
Нелинейность только одна,
выразительная сила меньше

GoogLeNet (Inception v1)



Convolution
Pooling
Softmax
Other

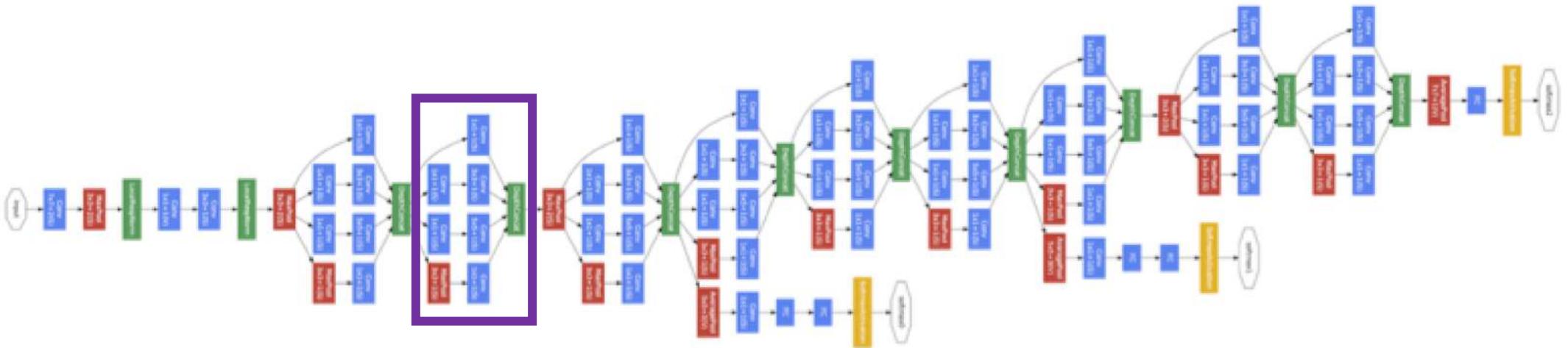
GoogLeNet (Inception v1)



Ранние выходы: идея была уменьшить затухание градиентов, но на практике это не очень помогло

Convolution
Pooling
Softmax
Other

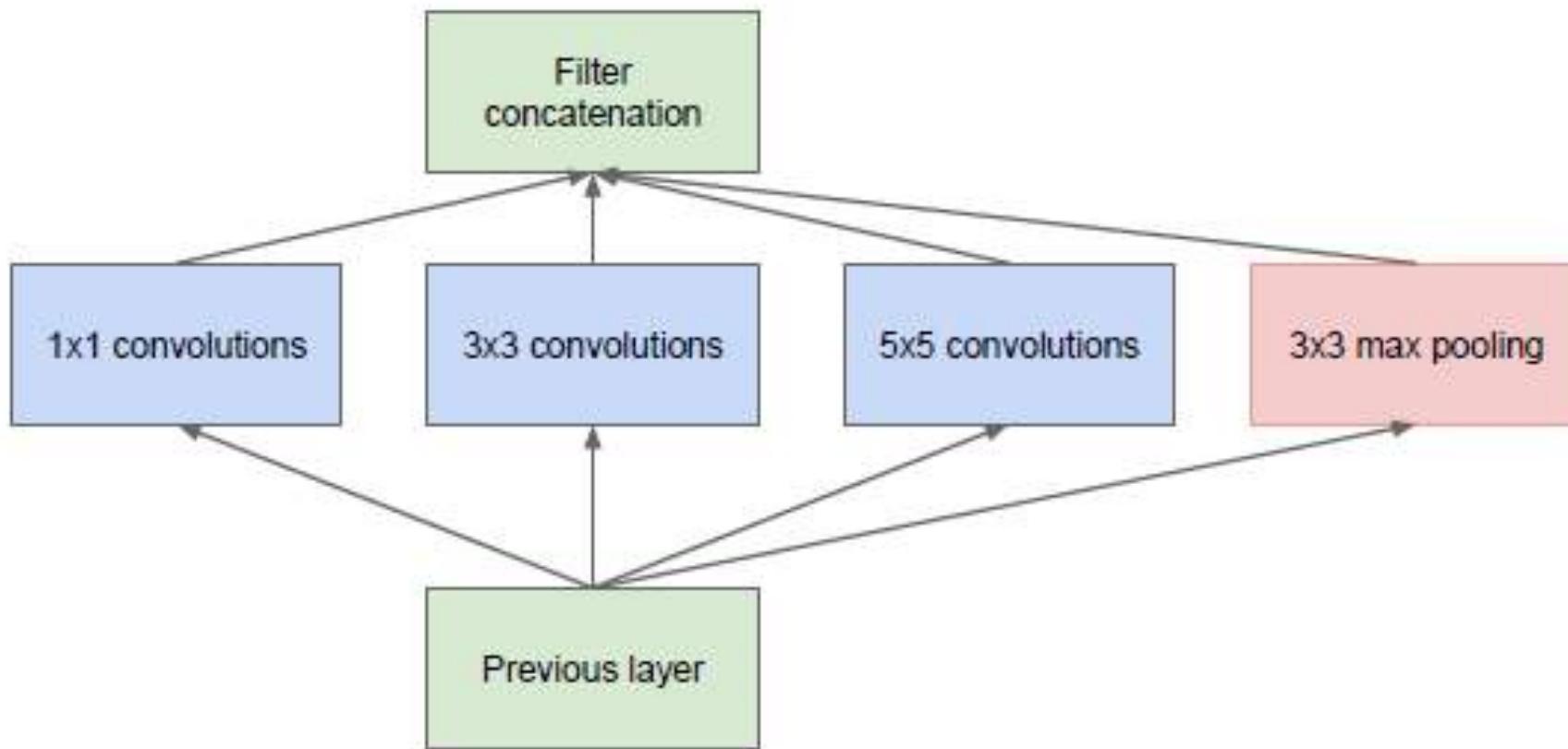
GoogLeNet (Inception v1)



Inception module

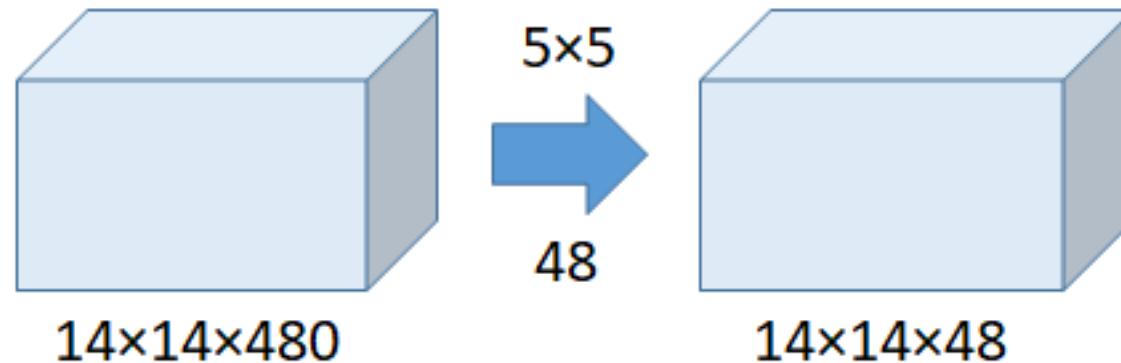
Convolution
Pooling
Softmax
Other

Inception-модуль (наивная версия)



(a) Inception module, naïve version

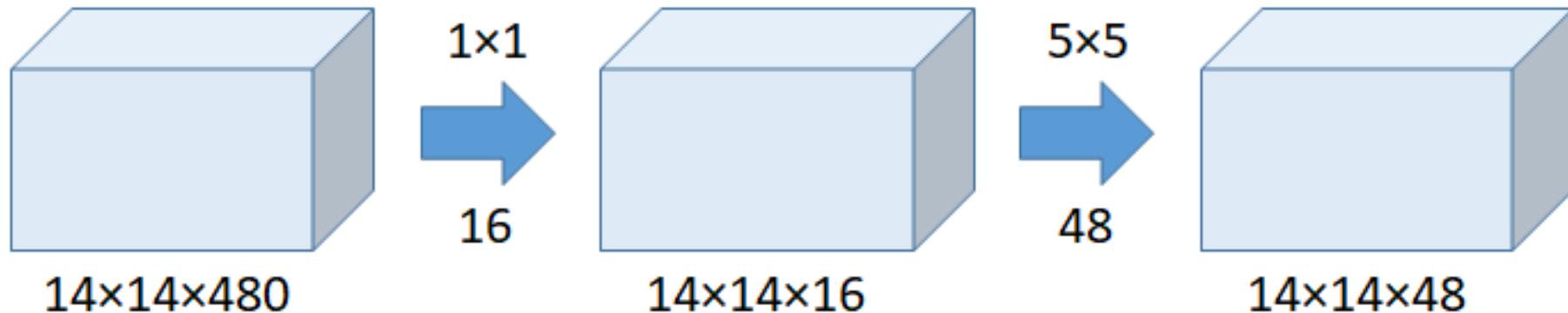
Проблема больших свёрток



14×14 раз прикладываем фильтр 5×5 ($5 \times 5 \times 480$ умножений и сложений с учетом глубины фильтра), повторяем для каждого из 48 фильтров:

$$14 \times 14 \times 5 \times 5 \times 480 \times 48 = \text{112.9 М операций}$$

Идея: свёртки 1x1

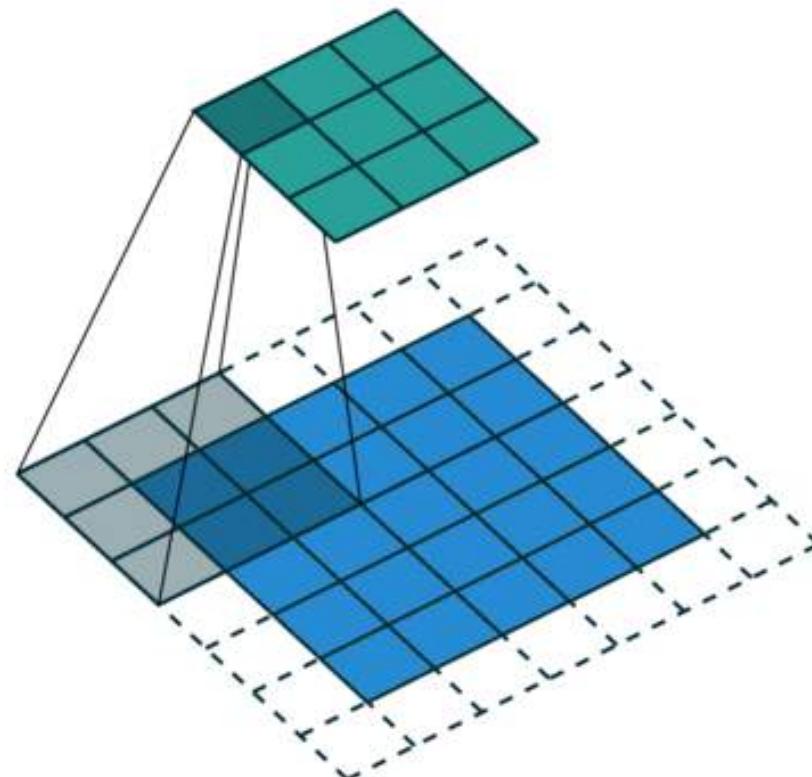


Применим 16 свёрток 1×1 (как обычно, той же глубины) –
снизим глубину с 480 на 16. Будет 5.3 М операций.

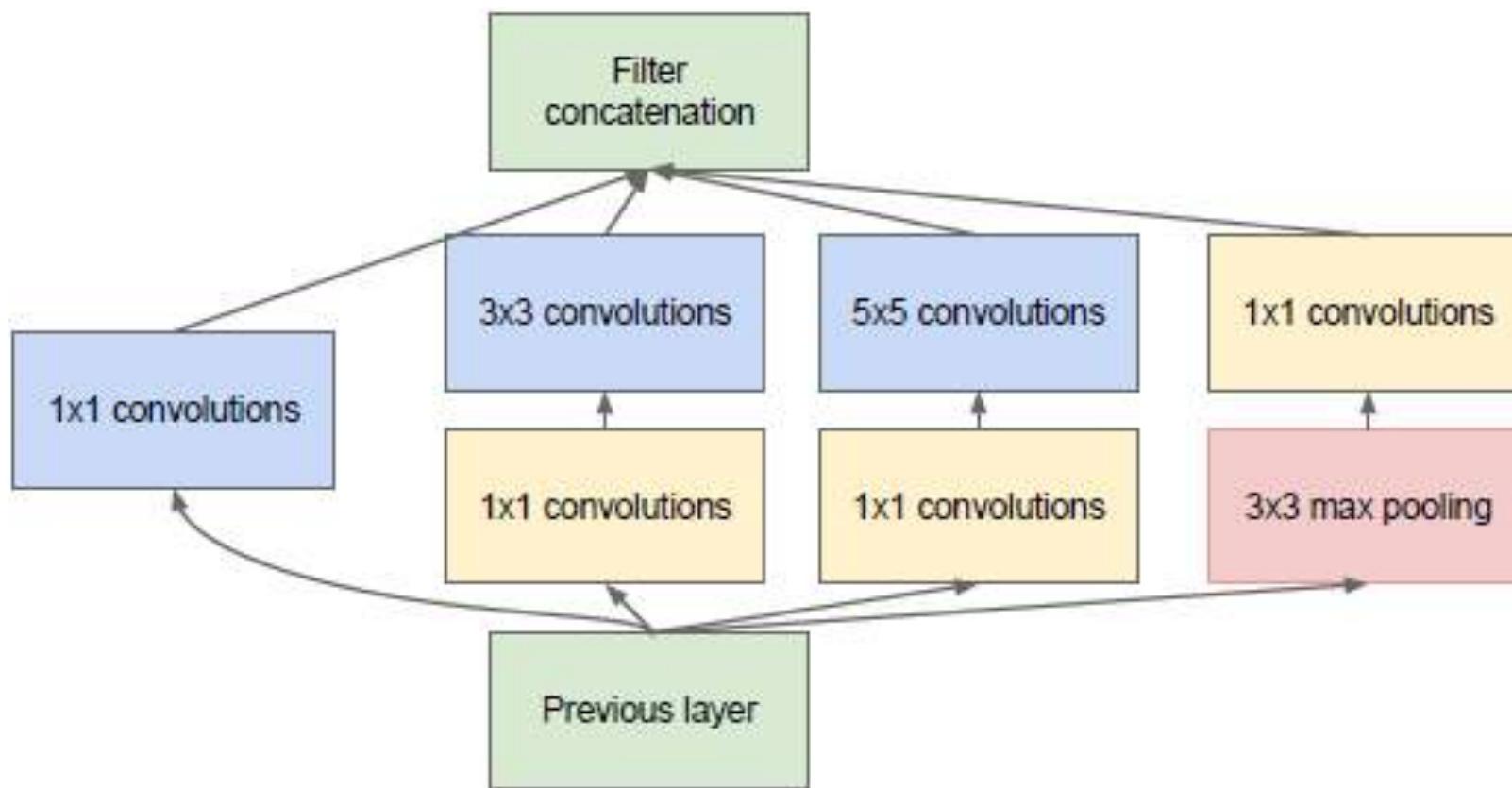
Таким образом, ширину/высоту можно уменьшать
пулингом, а глубину – свёртками 1×1

Небольшое отступление про пулинг

На самом деле для уменьшения ширины и высоты изображения не обязательно делать пулинг - можно продолжать делать свёртку, но со страйдом (шагом) больше единицы:

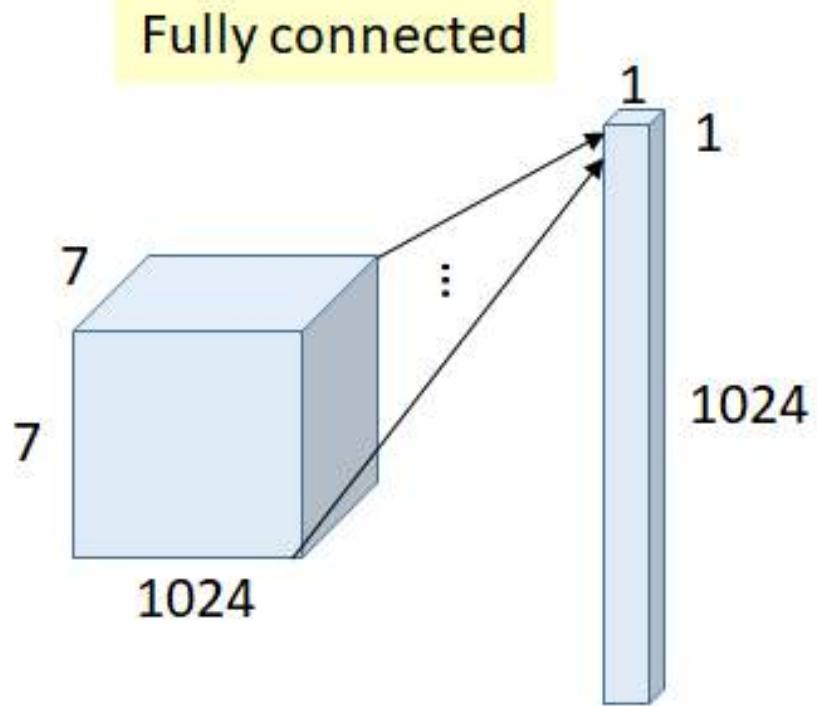


Inception-модуль

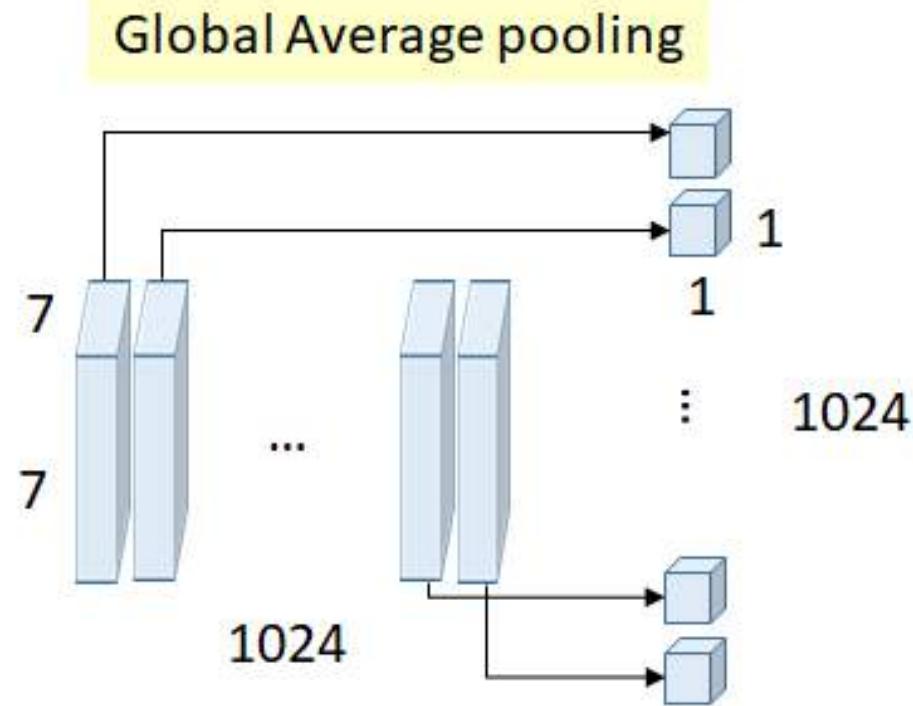


(b) Inception module with dimensionality reduction

Global Average Pooling



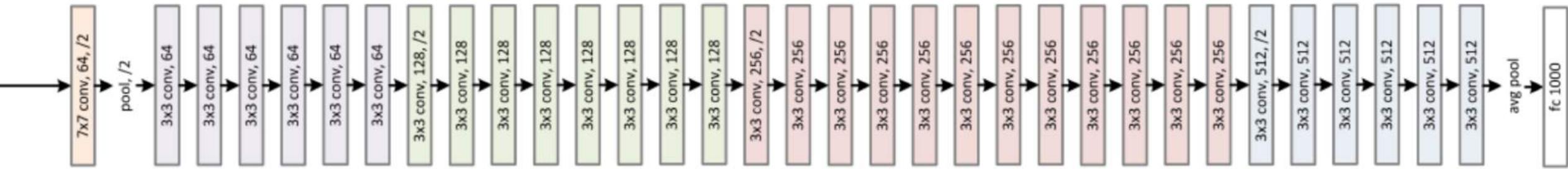
Применяли на выходах ранее,
еще +51 М весов



Опробовали в Inception:
+0 весов и +0.6% top-1 accuracy
на ImageNet

ResNet (Residual Net)

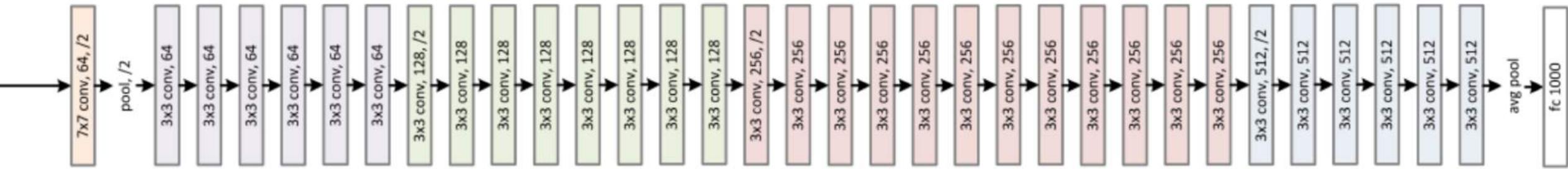
Попробуем сделать очень много слоев:



Ясно ли, в чем будет проблема?

ResNet (Residual Net)

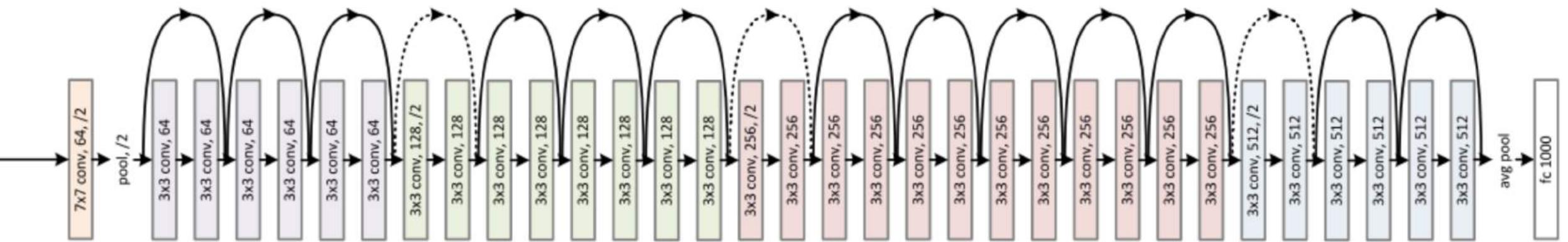
Попробуем сделать очень много слоев:



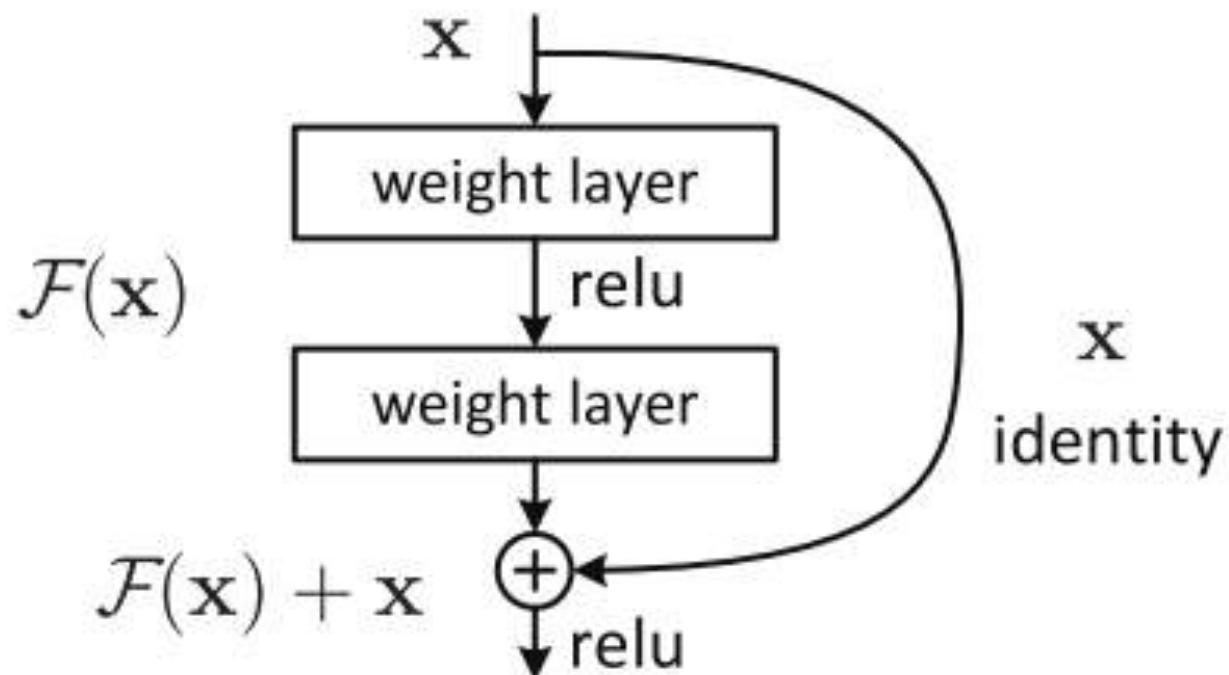
Ясно ли, в чем будет проблема? Верно, затухание градиента.

ResNet (Residual Net)

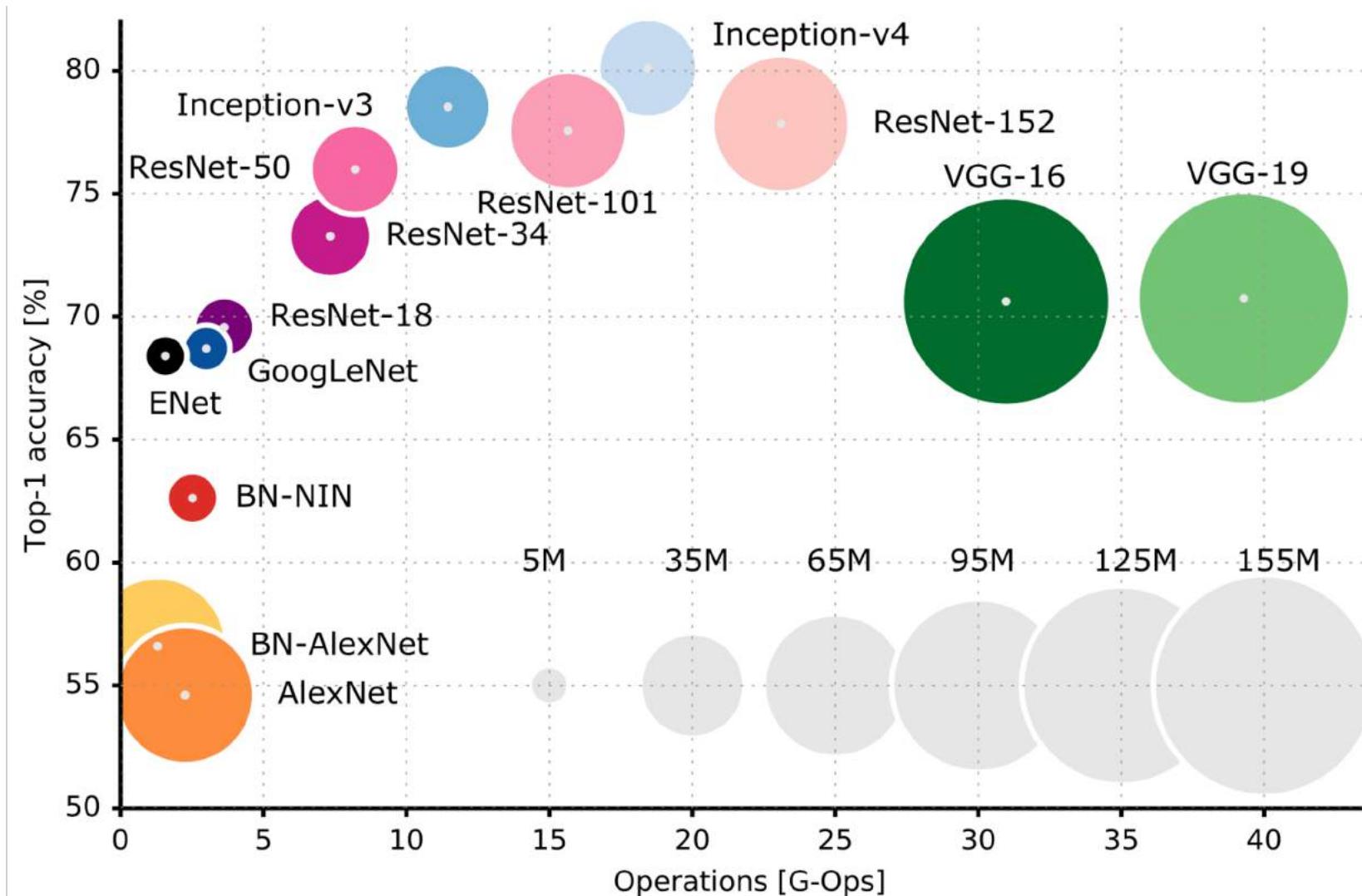
Но проблему решает добавление residual связей:



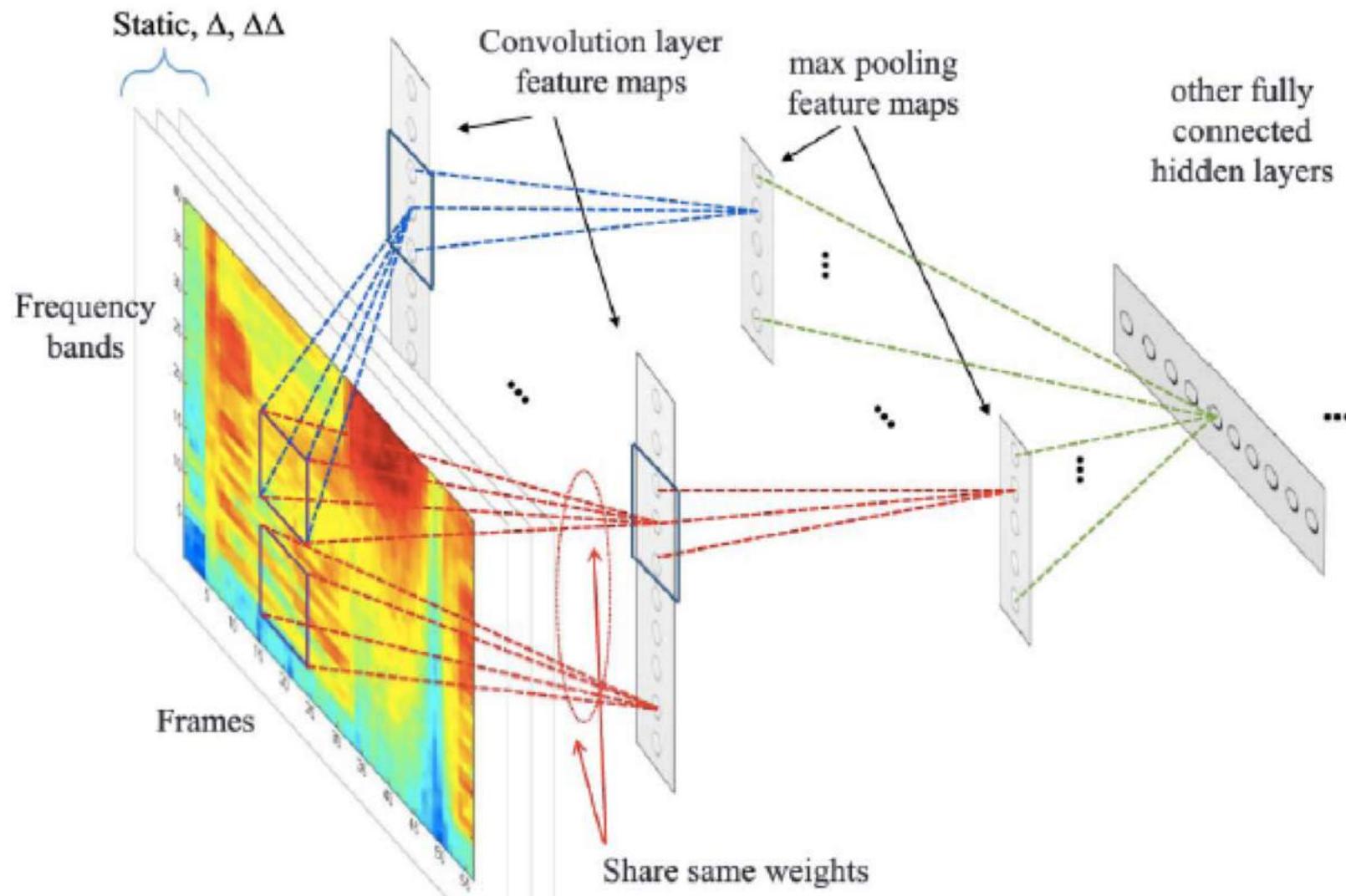
Residual-связь



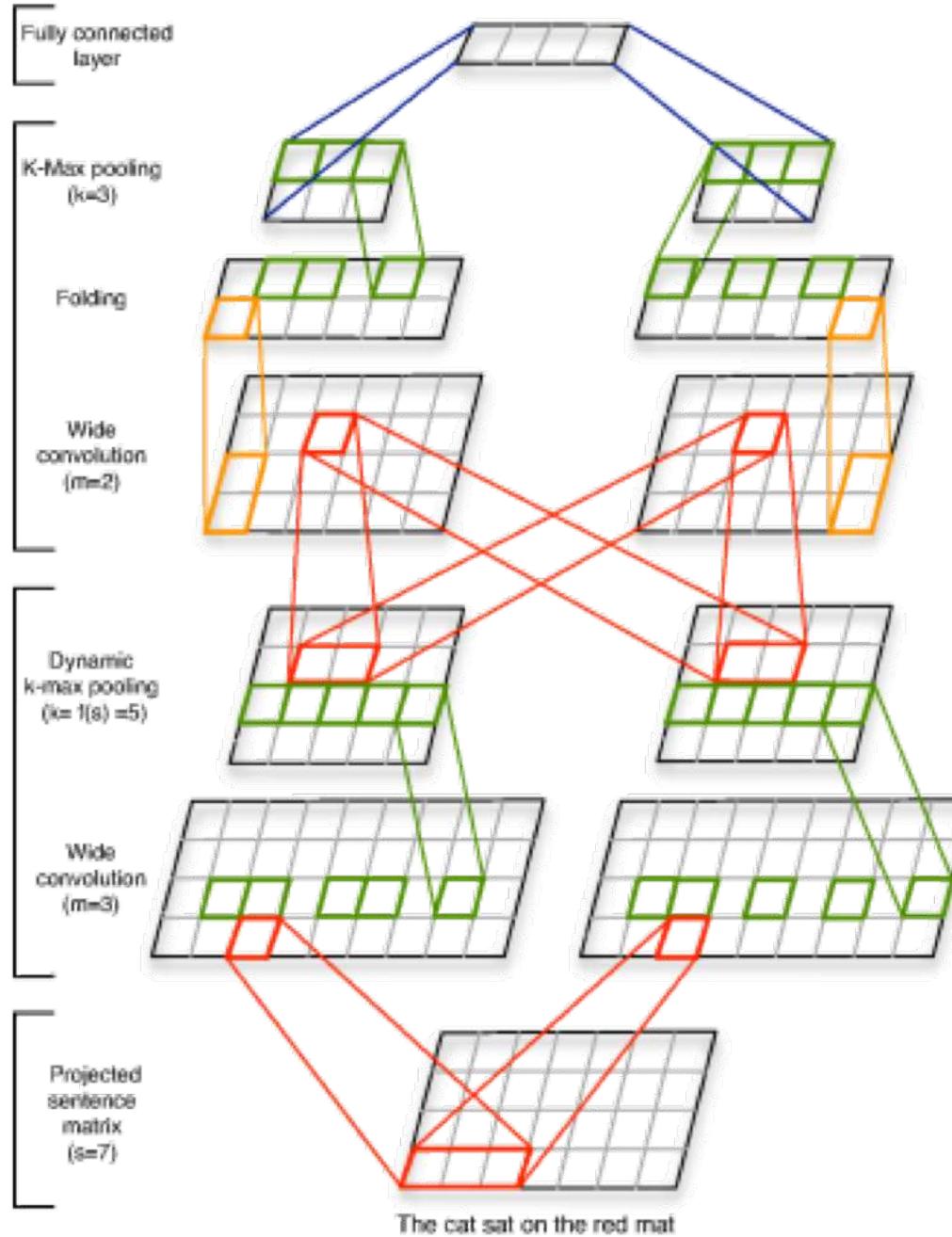
Сравнение архитектур



Распознавание речи



Сверточные нейросети в анализе текстов



План

1. Что такое нейросети
2. Обучение нейросетей
3. Свёрточные нейросети

Data Mining in Action

Лекция 7

Группа курса в Telegram:



<https://t.me/joinchat/B1OLTk74nRV56Dp1TDJGNA>