

Оптимизация моделей машииного обучения



План



Что и зачем мы пытаемся оптимизировать

План



Что и зачем мы пытаемся оптимизировать



Различные подходы для оптимизации больших моделей

План



Что и зачем мы пытаемся оптимизировать



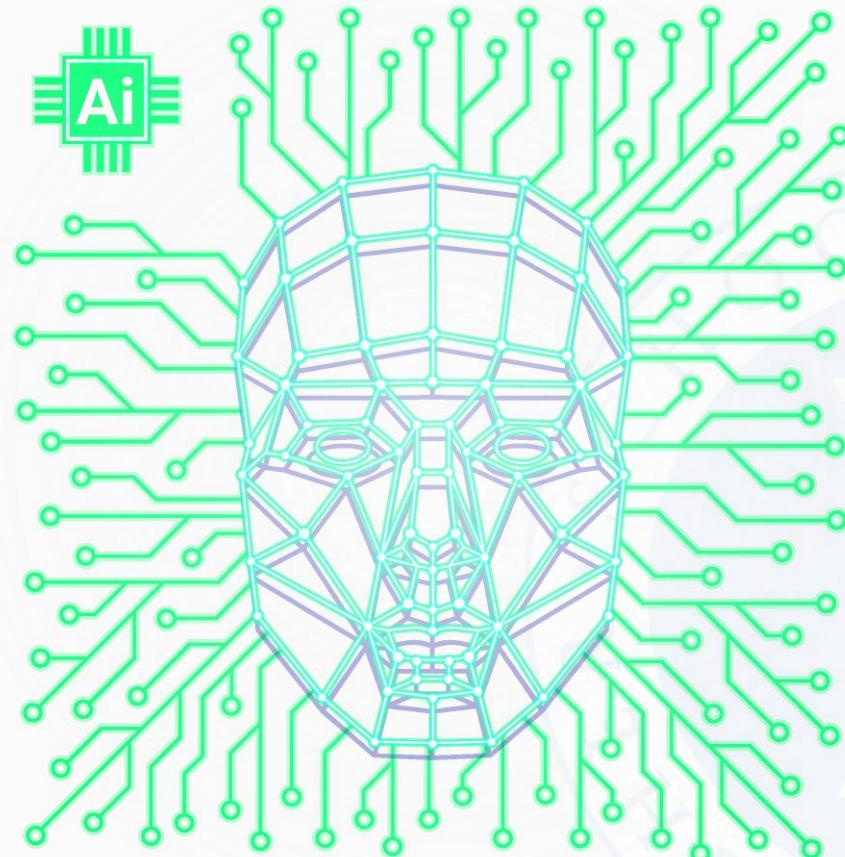
Различные подходы для оптимизации больших моделей



Практическое использование оптимизированных моделей на устройствах пользователей

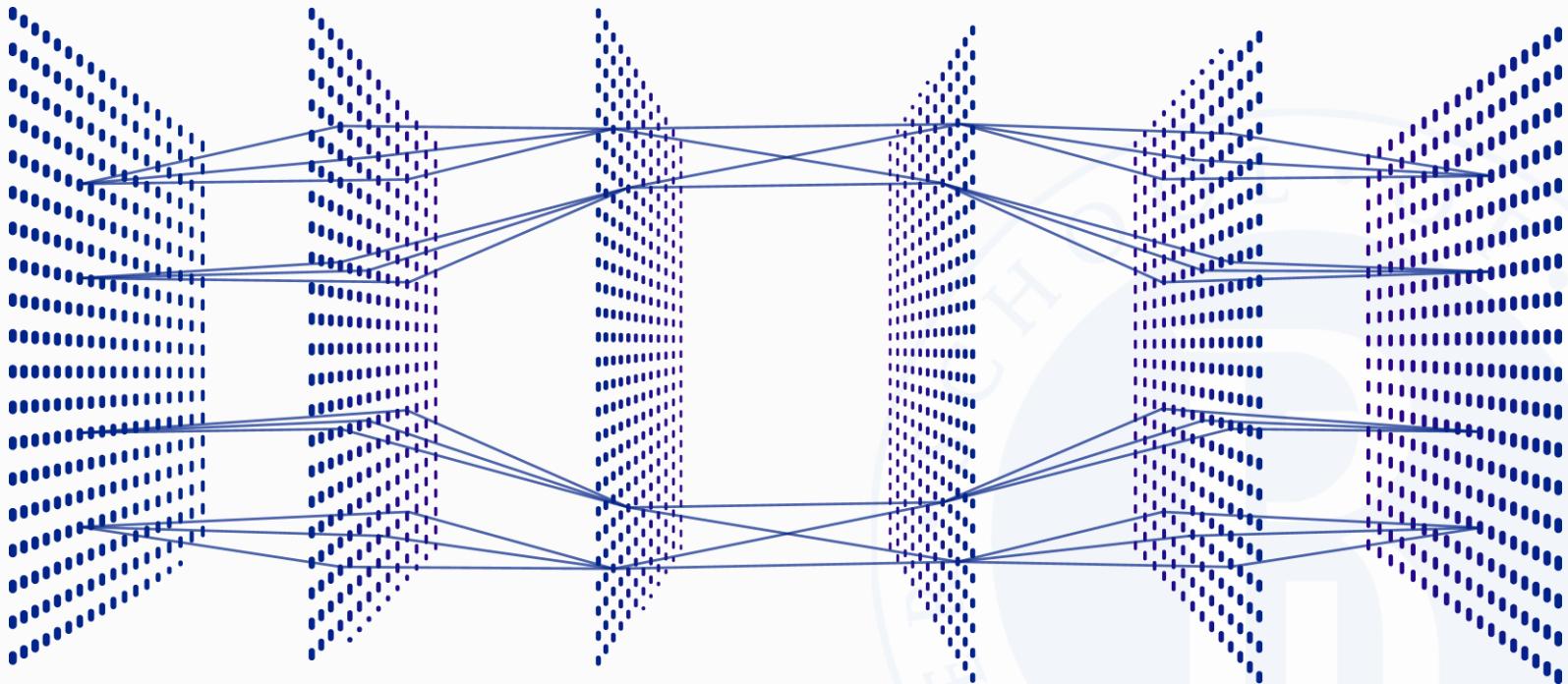
Большие модели для больших данных

→ Чтобы выявлять сложные зависимости в потоке данных часто требуются сложные модели



Большие модели для больших данных

→ Среди всех типов моделей наиболее сложными по устройству являются **нейронные сети**



Большие модели для больших данных

→ Примерное количество параметров в разных моделях:

- ResNet-50 — 23 миллиона
- VGGNet — 138 миллионов
- BERT — 340 миллионов
- GPT-2 — 1,5 миллиарда
- GPT-3 — 175 миллиардов



Большие модели для больших данных

→ Примерное количество параметров в разных моделях:

- ResNet-50 — 23 миллиона
- VGGNet — 138 миллионов
- BERT — 340 миллионов
- GPT-2 — 1,5 миллиарда
- GPT-3 — 175 миллиардов



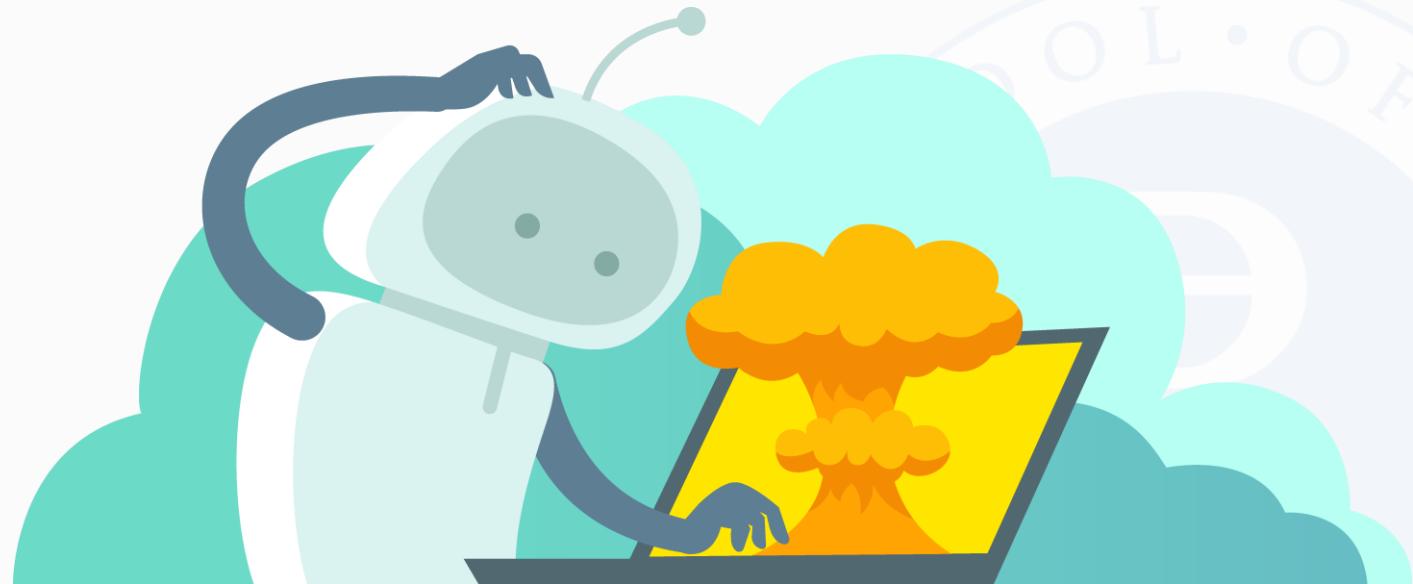
→ Интересный факт: в голове человека насчитывают всего от 20 до 80 миллиардов нейронов

Трудности с большими моделями



Работать с такими большими моделями тяжело:

- Их трудно обучать

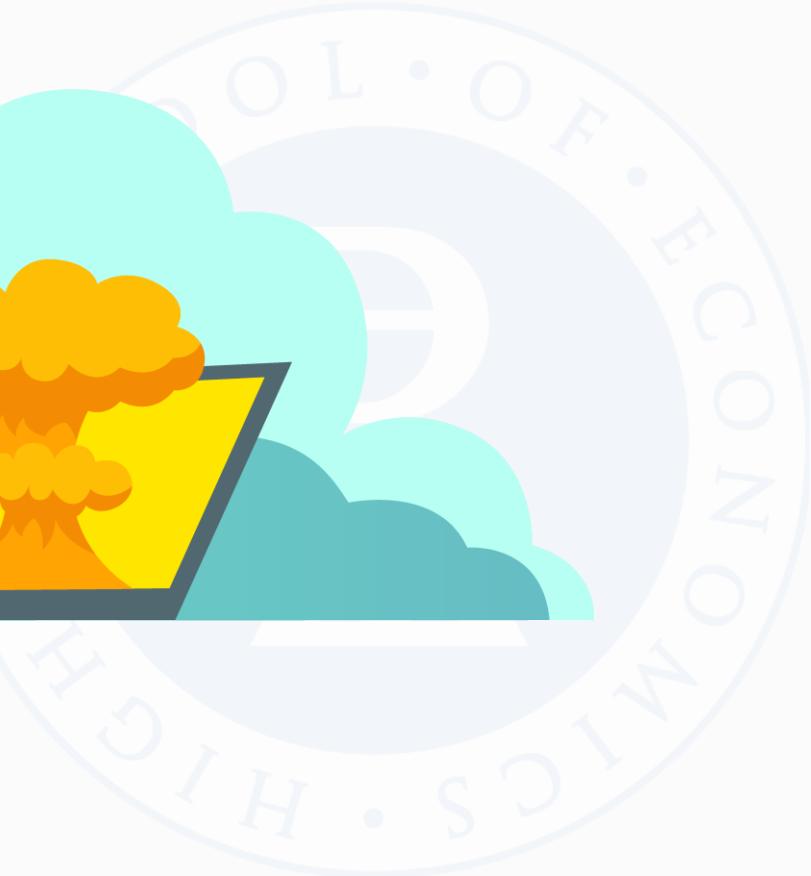
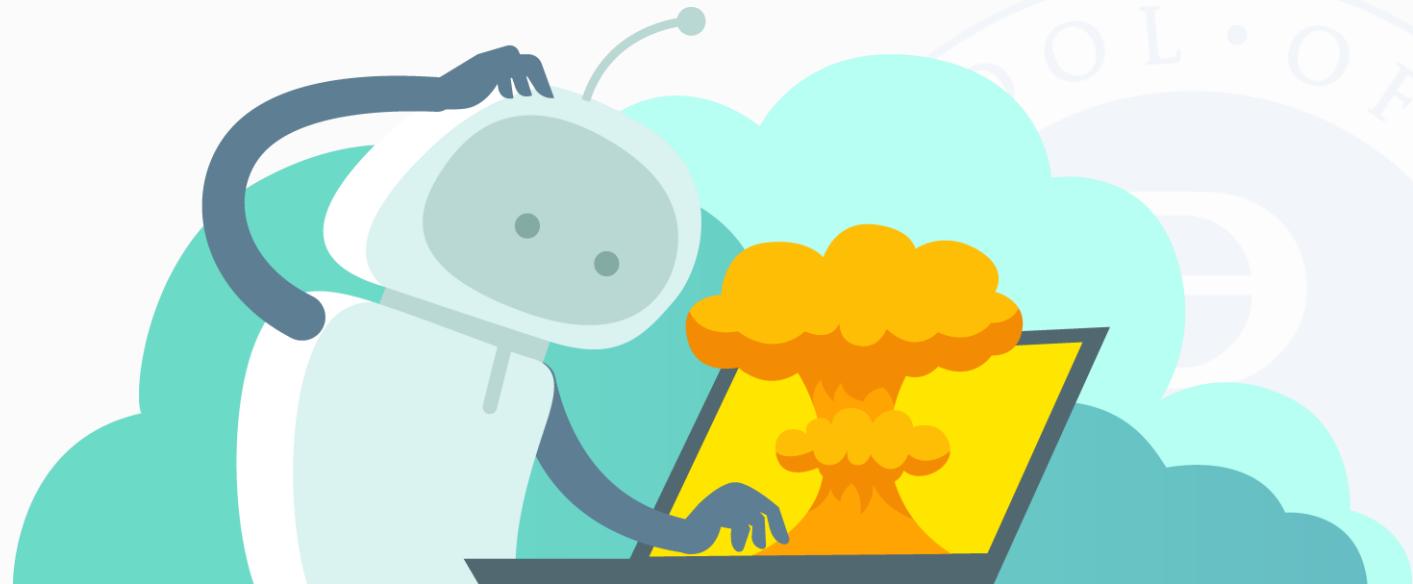


Трудности с большими моделями



Работать с такими большими моделями тяжело:

- Их трудно обучать
- Они занимают очень много места

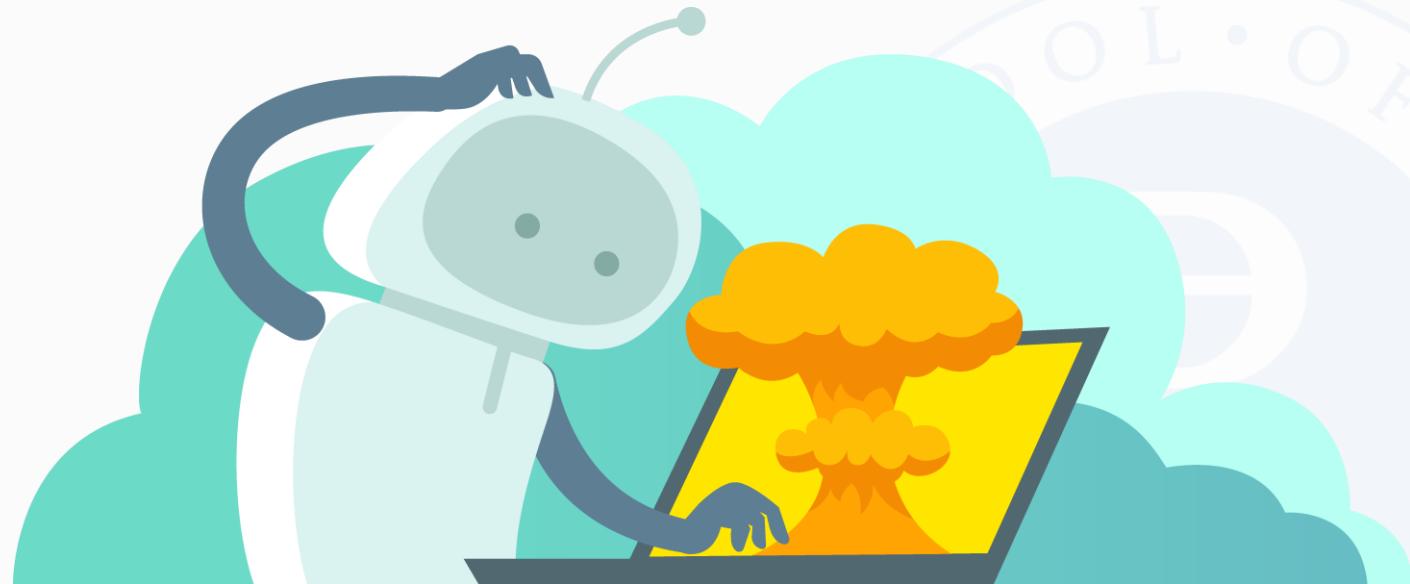


Трудности с большими моделями



Работать с такими большими моделями тяжело:

- Их трудно обучать
- Они занимают очень много места
- Они очень медленно вычисляют результат



Трудности с большими моделями

→ Многие современные задачи имеют высокие требования к алгоритмам, такие как:

- Хорошее качество результатов
- Высокая скорость работы
- Автономность — отсутствие постоянного интернет-соединения
- Ресурсоэффективность

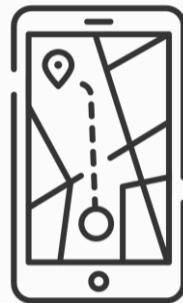


Трудности с большими моделями



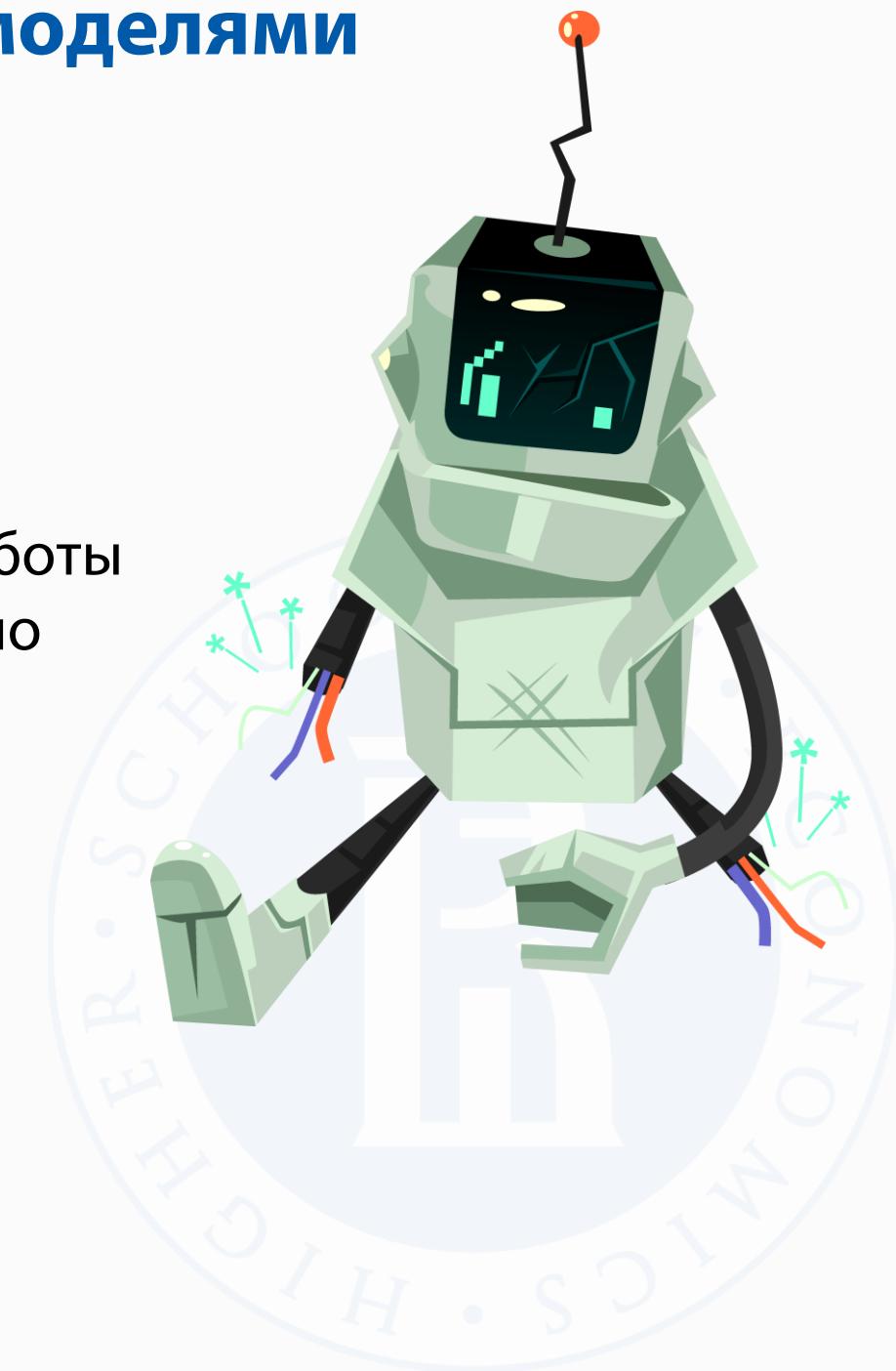
Примеры таких задач:

- Улучшение фотографий в телефоне
- Робот-пылесос
- Голосовой ассистент
- Автоматический перевод текстов
- Автокорректоры набора с клавиатуры
- Беспилотные автомобили



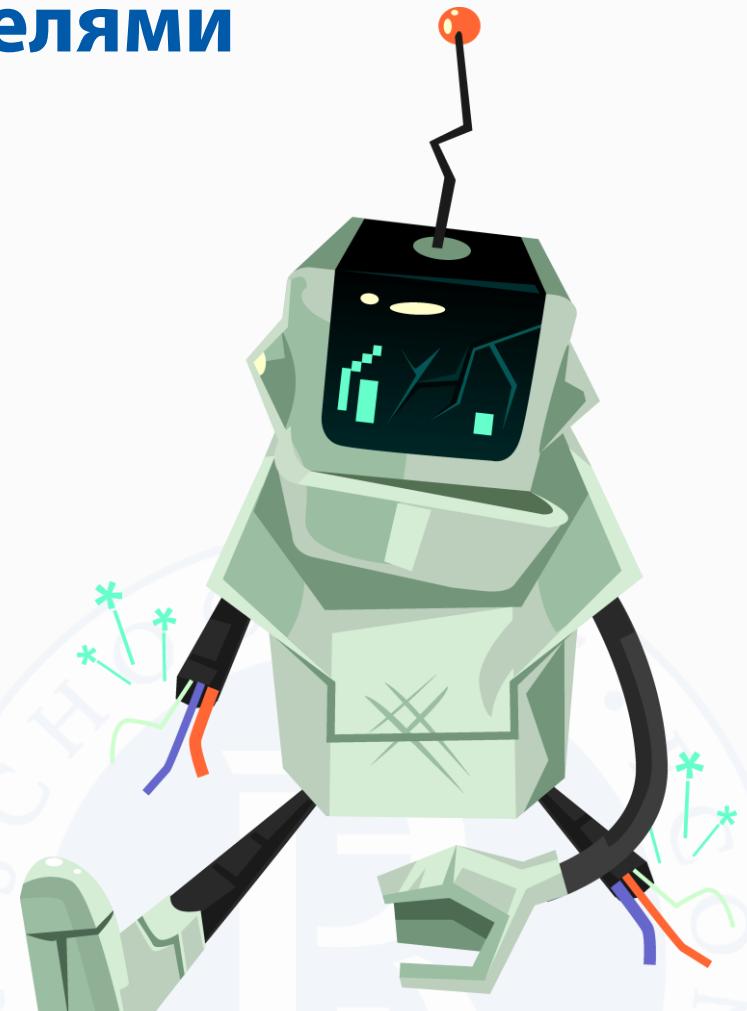
Трудности с большими моделями

- Можно пытаться запустить модели для этих задач на больших серверах
- Однако это может оказаться очень **дорого**, а скорость работы останется все еще достаточно **медленной**



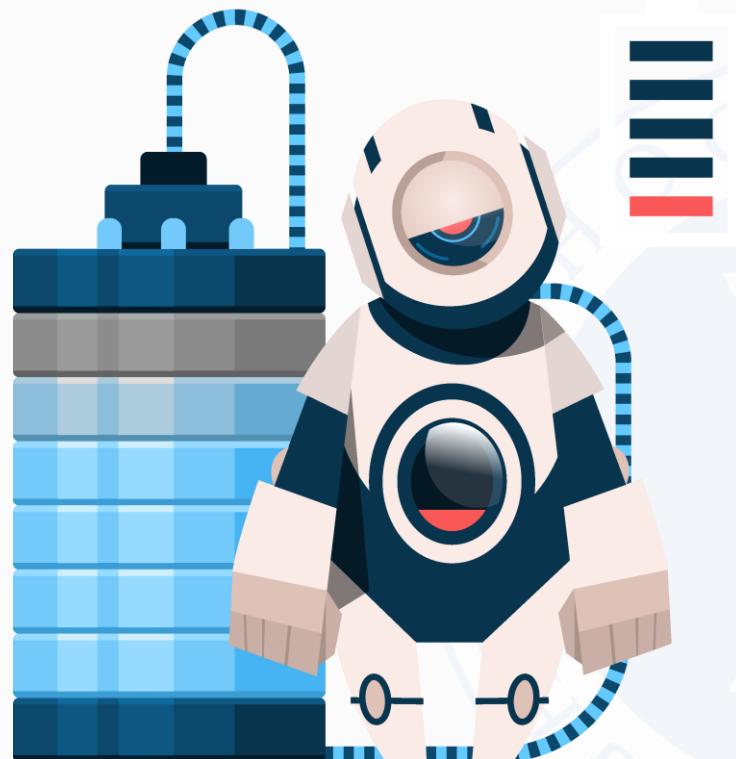
Трудности с большими моделями

- Можно пытаться запустить модели для этих задач на больших серверах
- Однако это может оказаться очень **дорого**, а скорость работы останется все еще достаточно **медленной**
- Более того, для работы устройствам будет требоваться **постоянный доступ в интернет**



Трудности с большими моделями

→ Если запускать модель на **самых устройствах**, то ей может не хватить ресурсов для запуска, **скорость** работы может оказаться слишком **низкой**, а потребление **батареи** устройства — слишком **высоким**



Оптимизация моделей

→ Таким образом, оптимизация моделей по размеру и скорости работы — важный элемент запуска алгоритма в реальный мир



Оптимизация моделей



На этой неделе узнаем про:

- Методы оптимизации нейронных сетей
- Выбор подходящей архитектуры
- Примеры запуска на устройстве пользователя

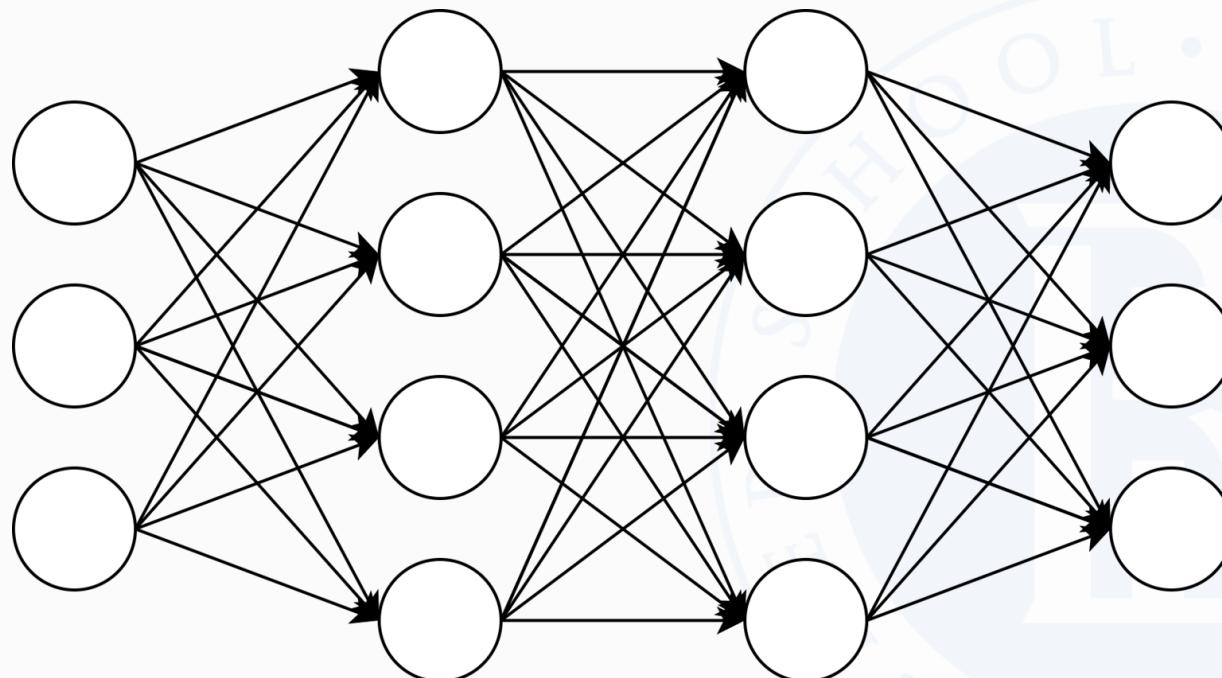


Прореживание сети



Убираем лишнее

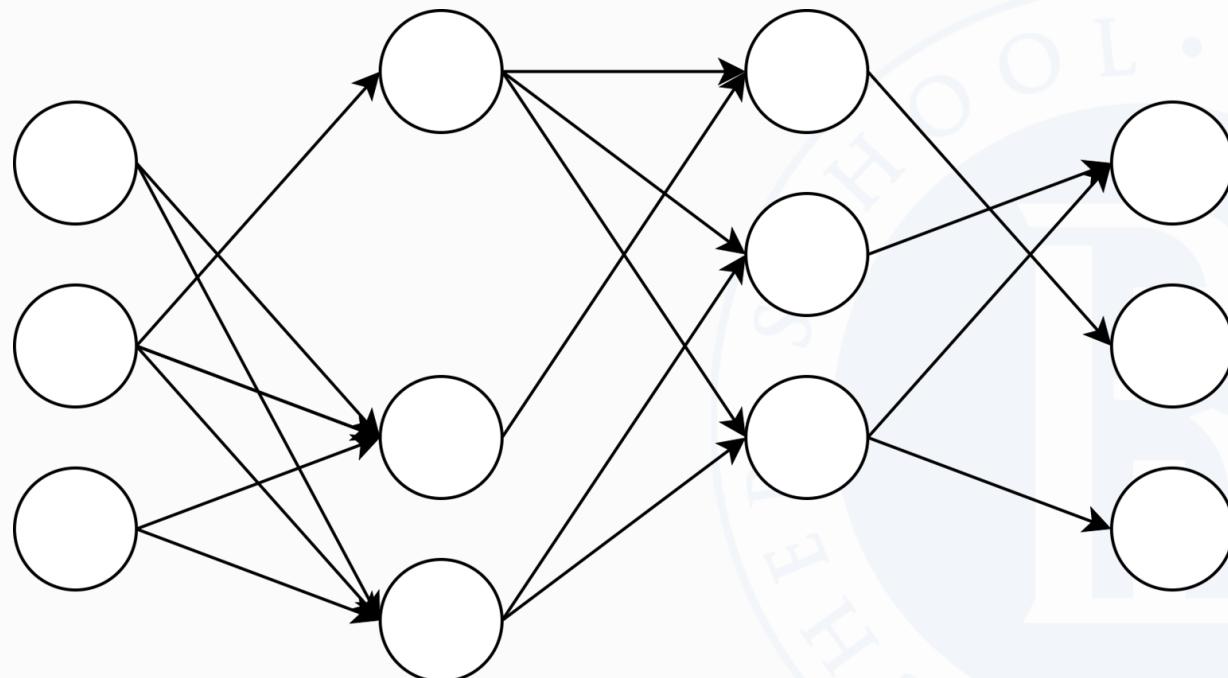
→ В глубоких сетях используется **огромное количество параметров**, которые настраиваются в процессе обучения



Убираем лишнее

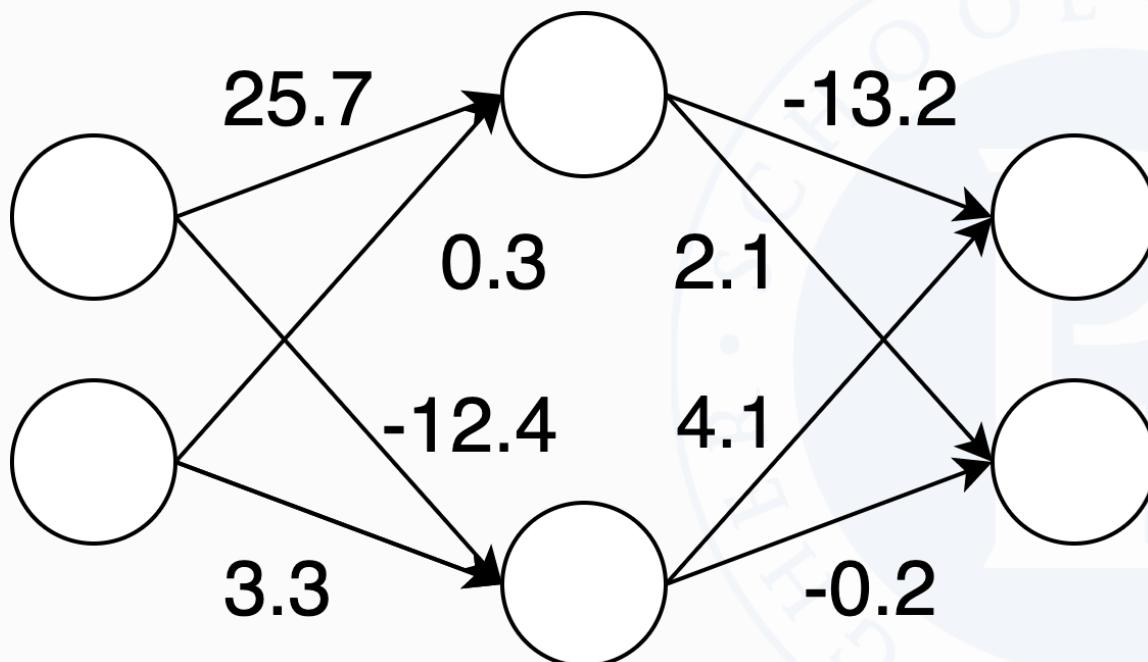
→ Практика показывает, что часто такое количество избыточно для поставленной задачи

→ Поэтому логичная идея — удалить лишние параметры



Итеративное прореживание

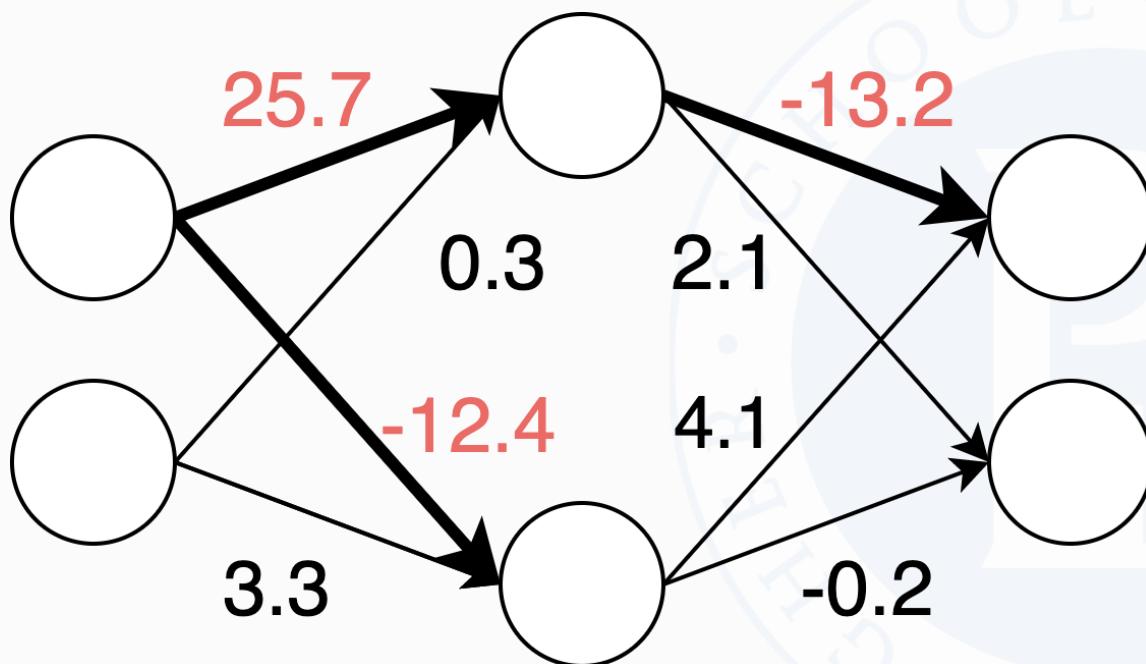
- Хочется научиться удалять именно **лишние** веса, а **не случайные**
- Один из подходов — смотреть на **модули** коэффициентов внутри уже обученной модели



Итеративное прореживание



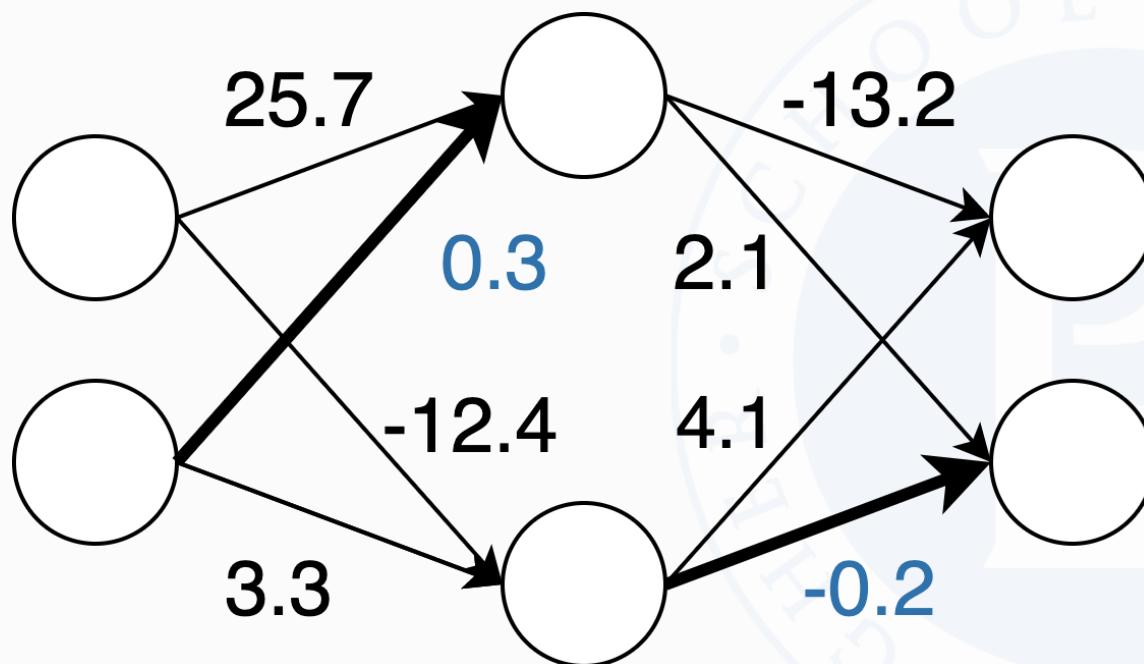
Если значение веса по модулю **большое**,
то удалять его, очевидно, **не стоит**



Итеративное прореживание

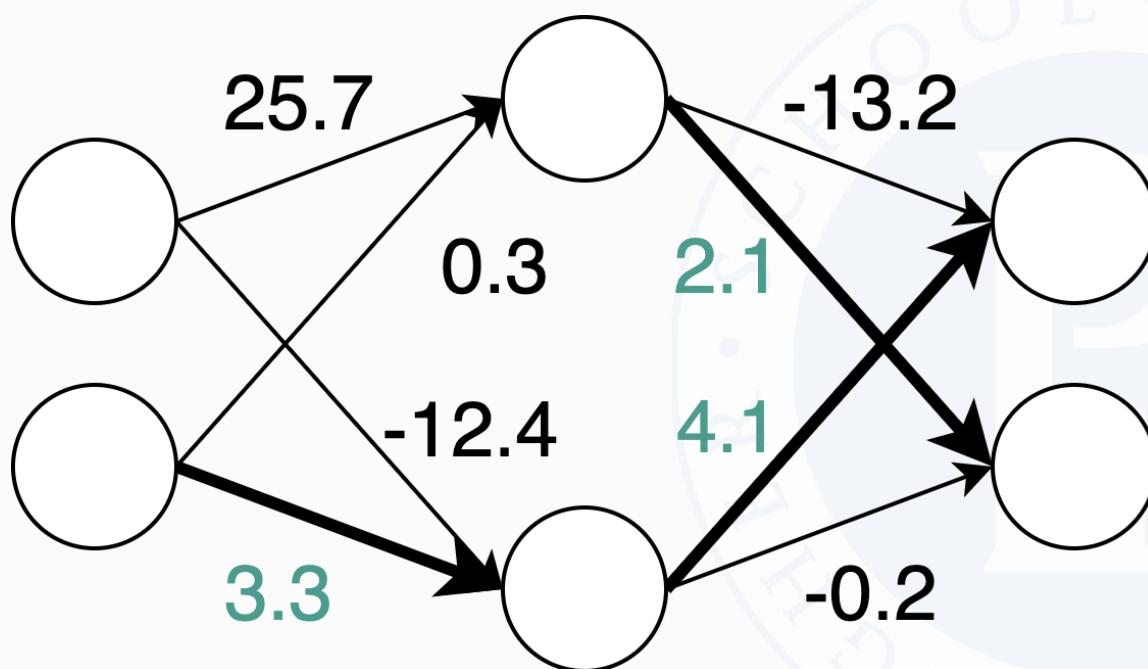


Если же оно **почти около нуля**, то можно ожидать,
что качество особо не изменится, если его **занулить**
(то же самое, что **удалить**)



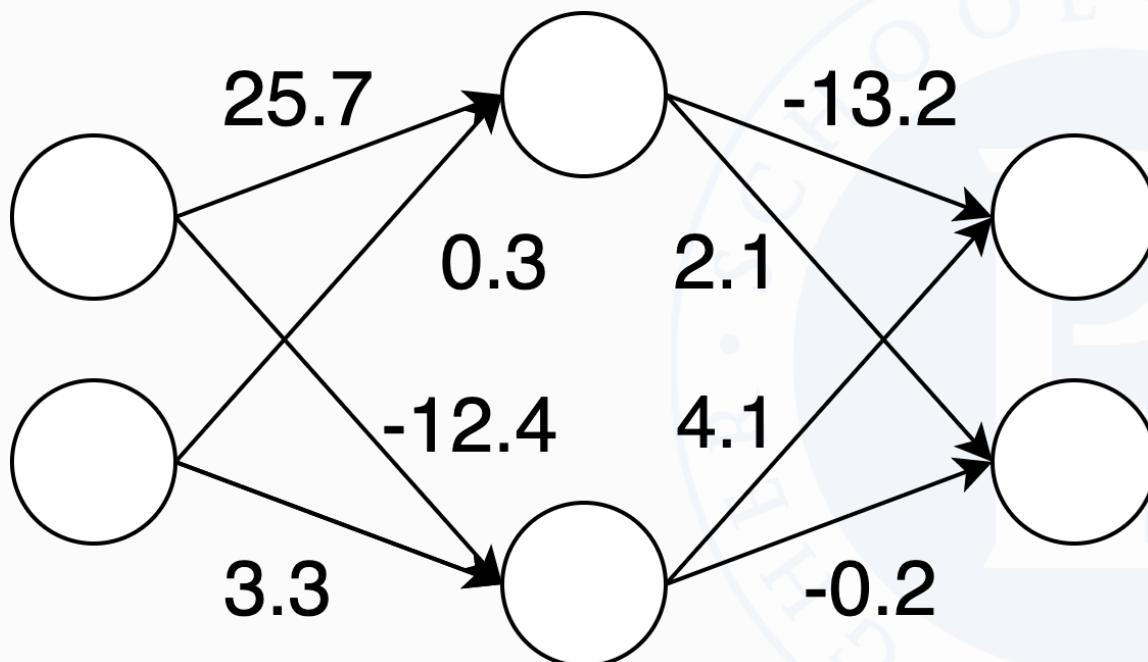
Итеративное прореживание

→ Про остальные веса **непонятно** — может быть, они важны, а может и нет



Итеративное прореживание

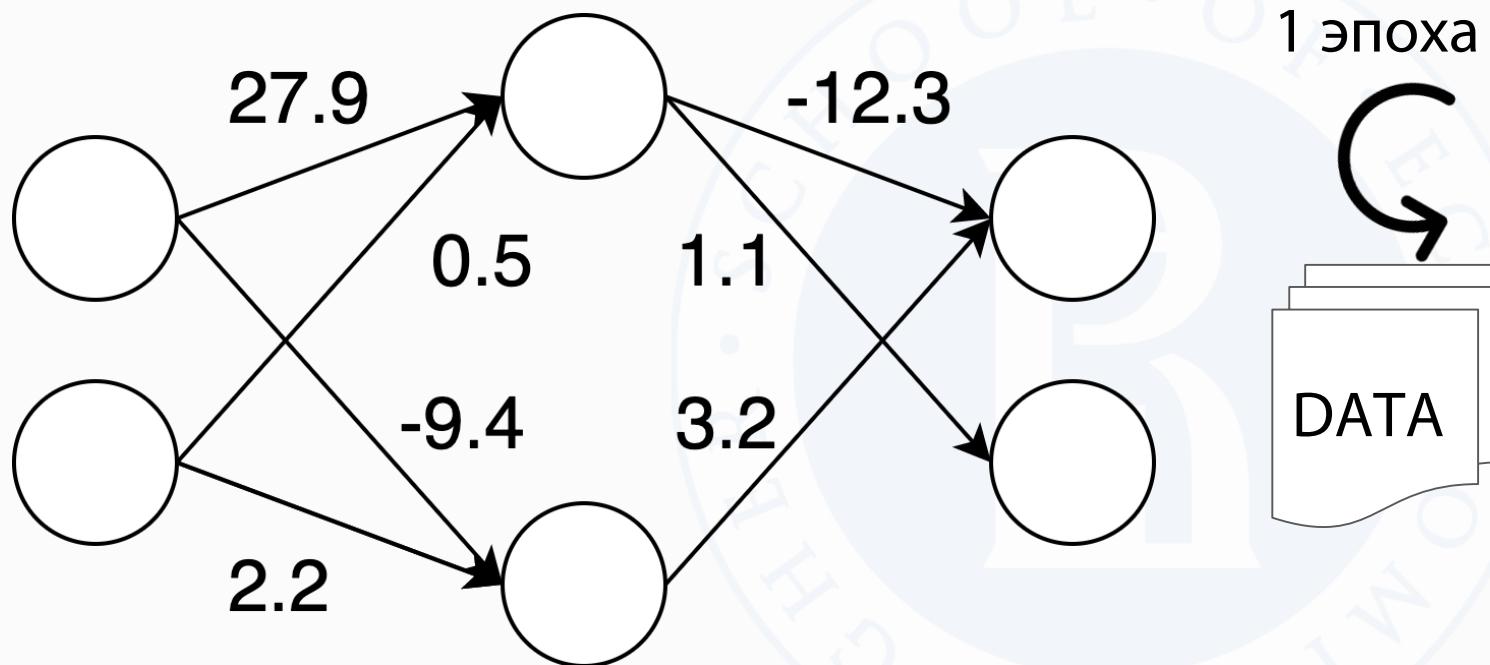
- Чтобы **не удалить лишнего**, будем отключать **небольшую** часть самых «неважных» связей
- В нашем примере отключим вначале связь с весом -0.2



Итеративное прореживание

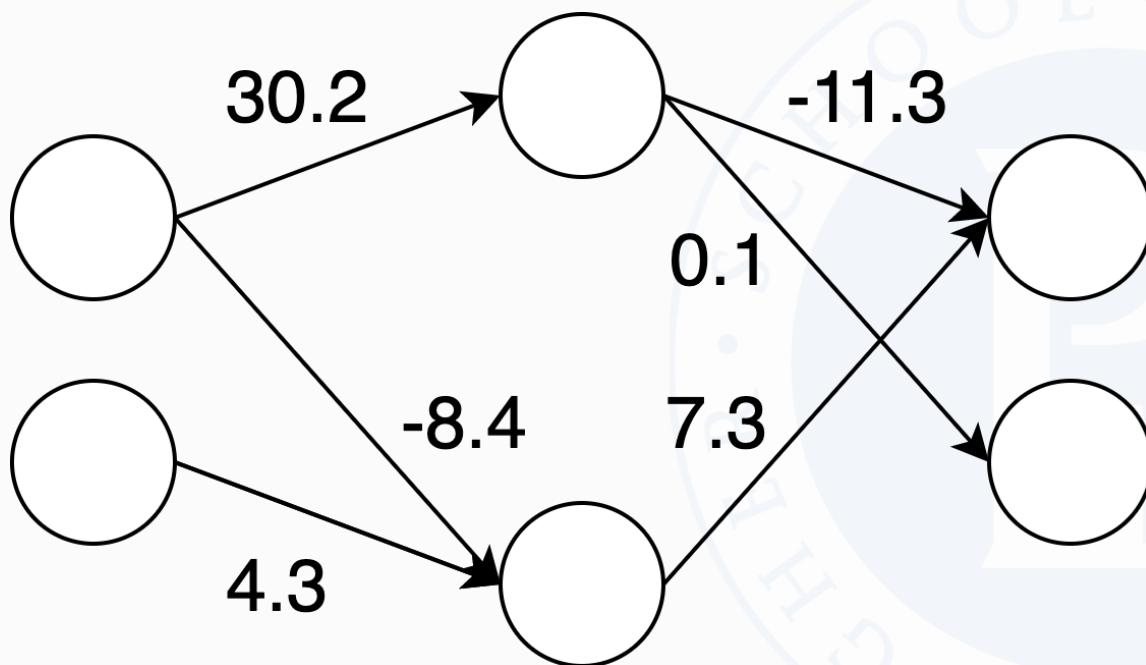


Чтобы компенсировать возможную потерю,
дообучим полученную модель в течение одной эпохи



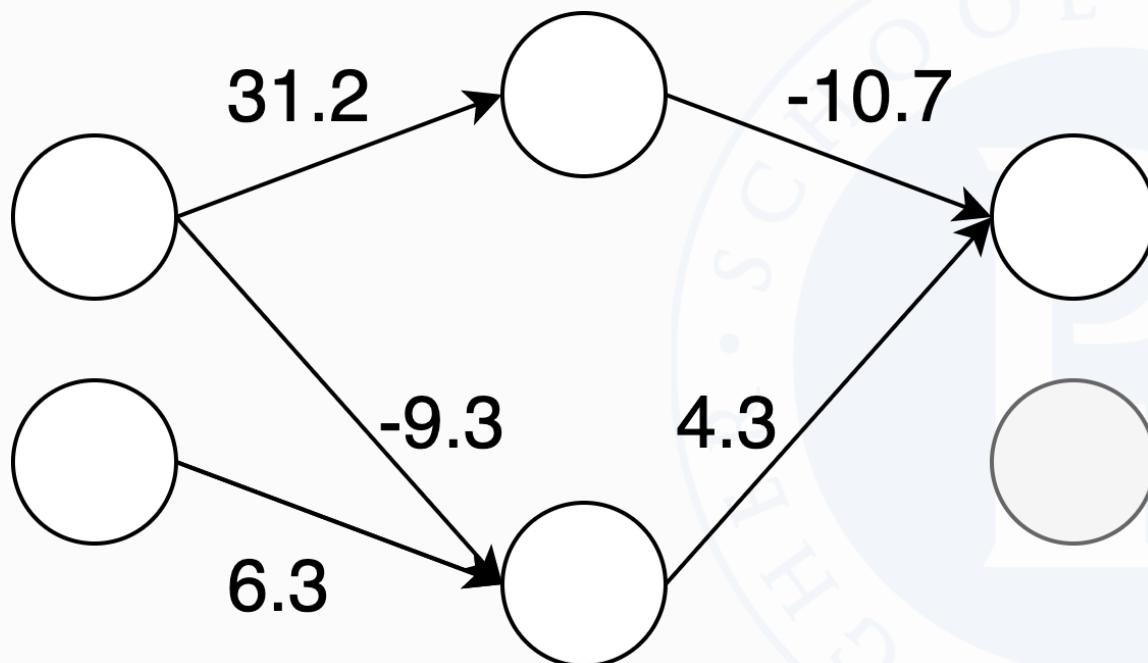
Итеративное прореживание

→ Повторим процедуру — удалим еще один вес
и еще раз дообучим



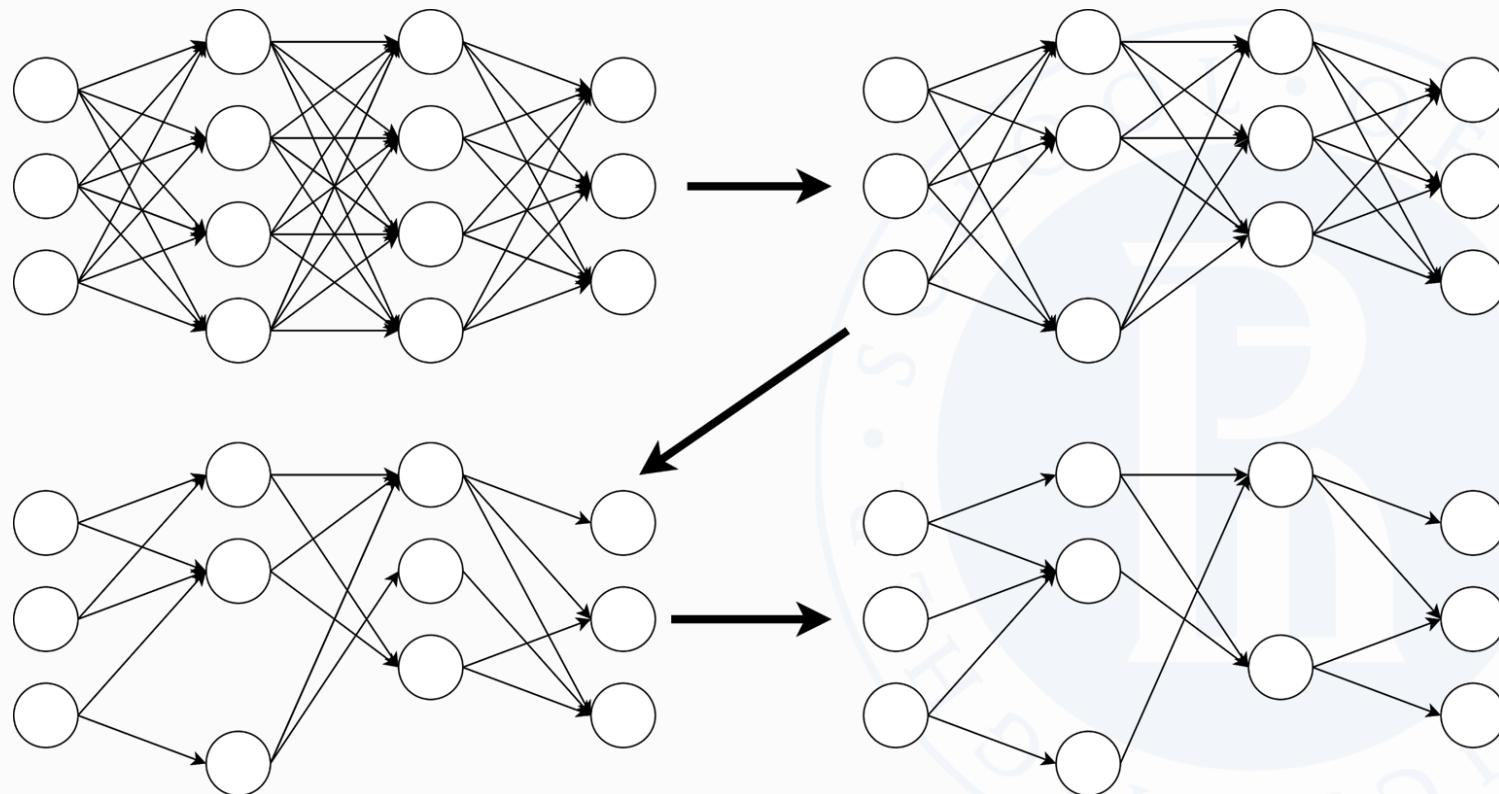
Итеративное прореживание

→ Повторяя процесс, мы надеемся, что оставшиеся важные связи **перехватят** задачи удаленных, а **неважные отключатся** окончательно. Можем удалять, пока качество не начнет падать



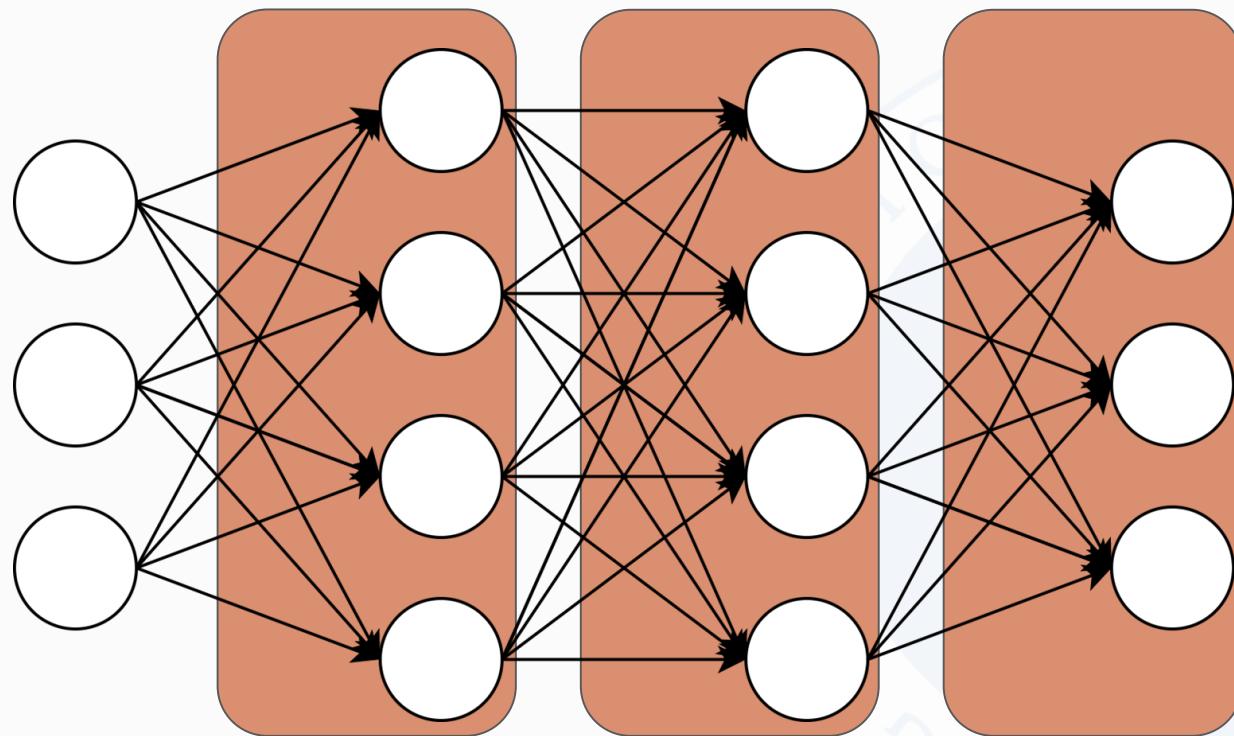
Итеративное прореживание

→ Процесс достаточно ресурсозатратный, поэтому удалять можно не по одному весу, а, например, по 10% сети за раз



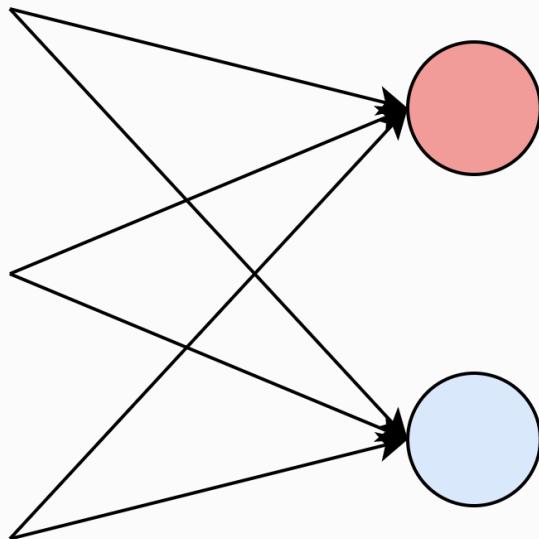
Вариации алгоритма

→ В текущую схему прореживания можно вносить различные модификации. Например, удалять веса **не глобально** по всей сети, а **отдельно по слоям**



Вариации алгоритма

→ Вместо **отдельных связей** можно смотреть на **нейроны**.
Значимость нейрона — это **совокупная значимость**
входящих связей. Ее можно считать по-разному.
Например, **L2** норма (сумма квадратов весов)



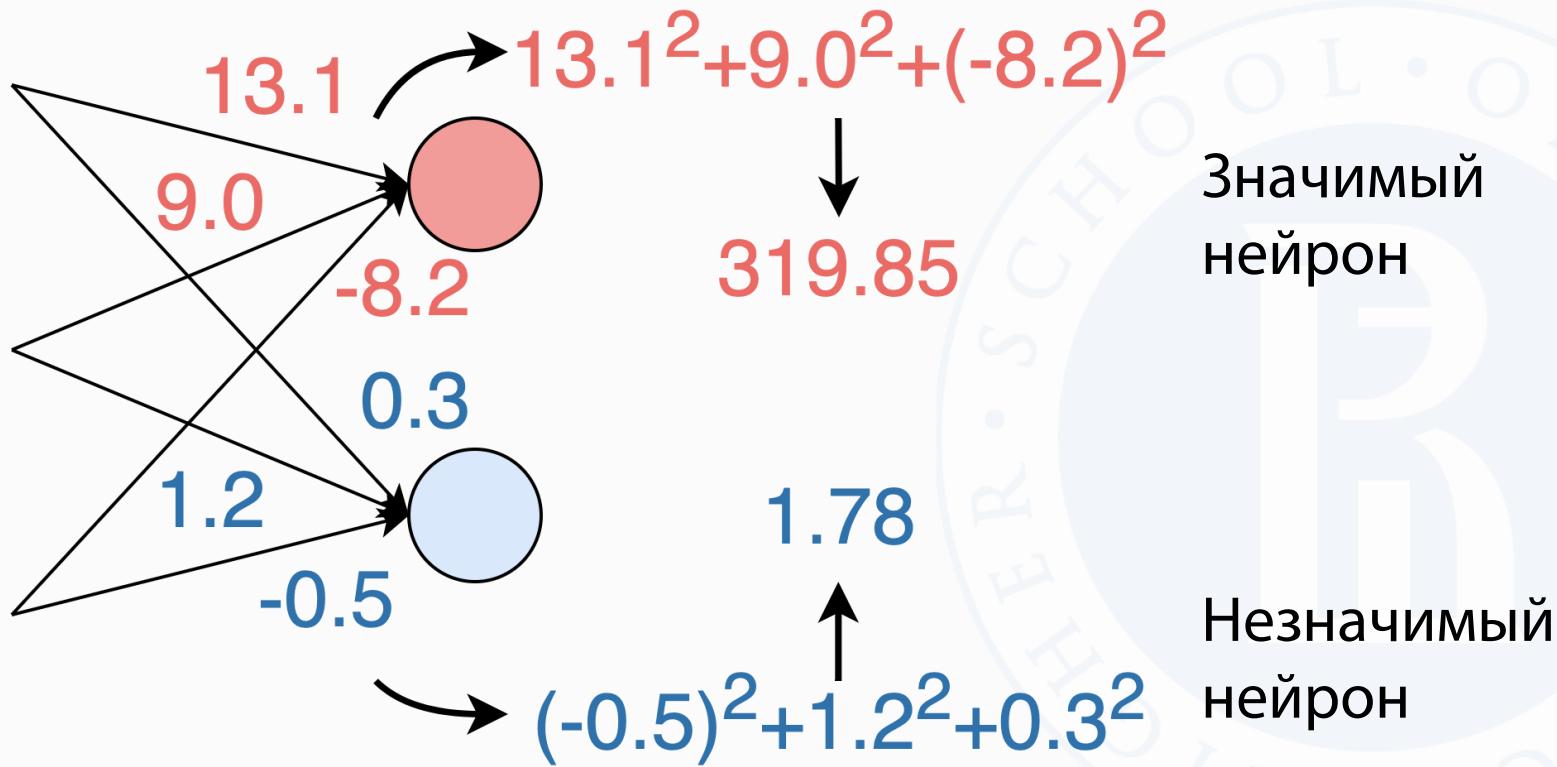
Вариации алгоритма



Вместо **отдельных связей** можно смотреть на **нейроны**.

Значимость нейрона — это **совокупная значимость** входящих связей. Ее можно считать по-разному.

Например, **L2** норма (сумма квадратов весов)

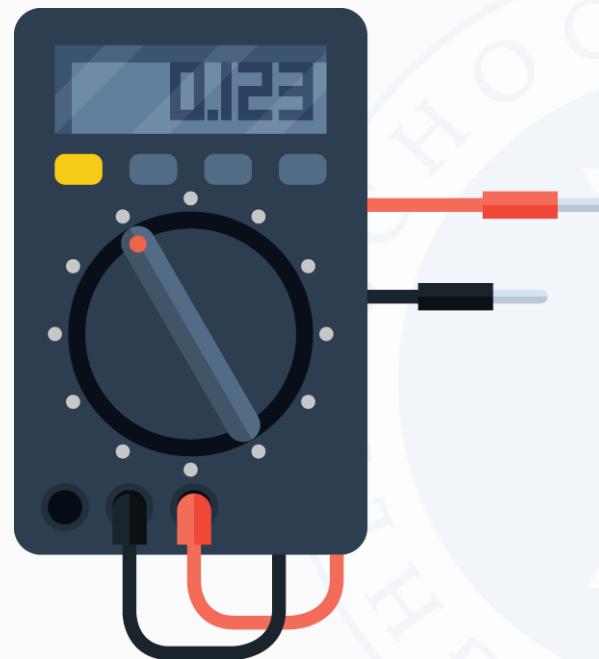


Итеративное прореживание

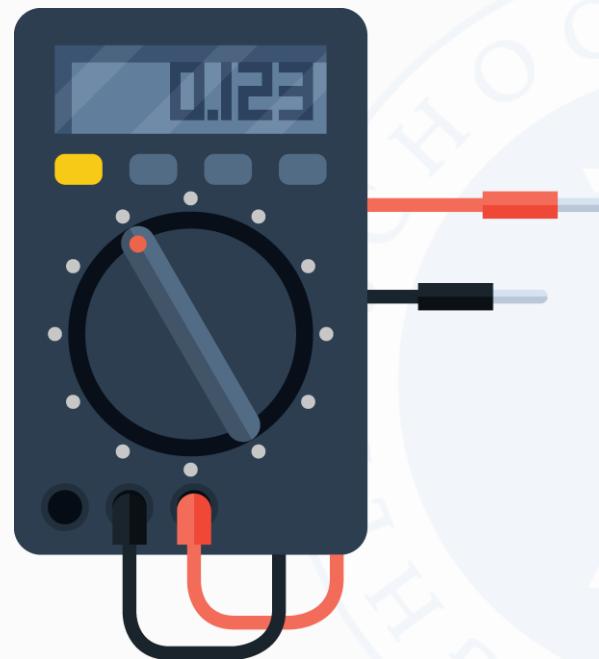
- Прием позволяет сети более «осознанно» решать поставленную задачу, убирая **избыточные** элементы из ее структуры



→ Используя [простые](#) критерии «важности связи», можно получать не самые оптимальные архитектуры

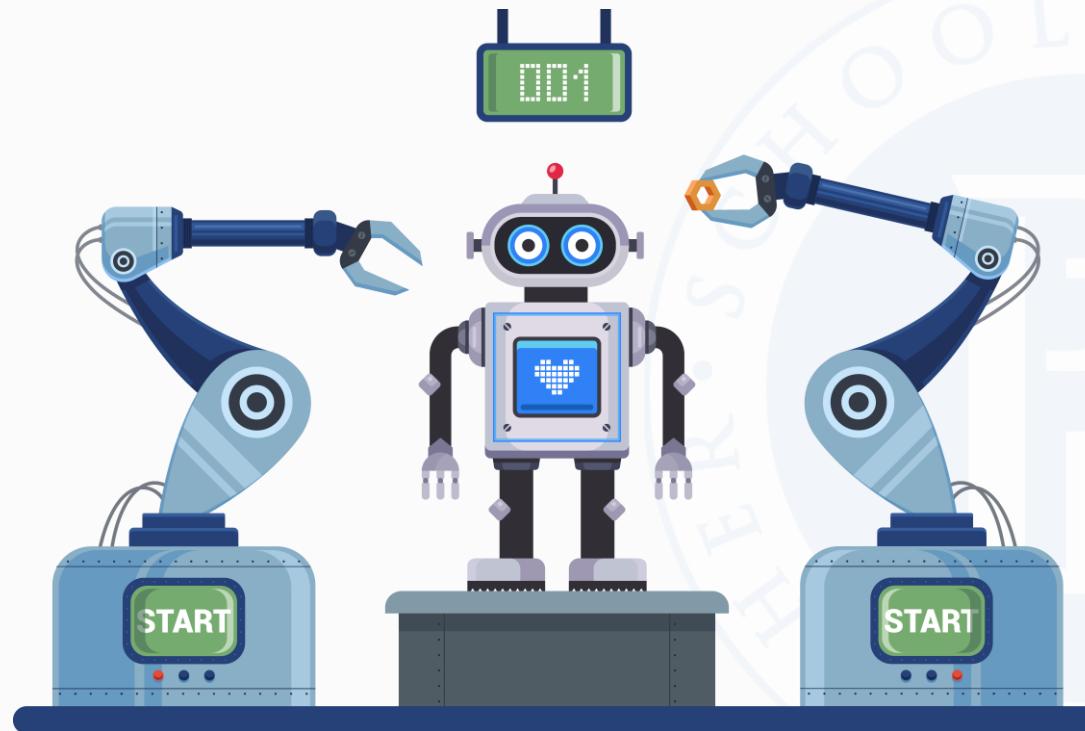


- Используя **простые** критерии «**важности связи**», можно получать не самые оптимальные архитектуры
- Поэтому возникает идея более **умного** оценивания характеристики сжатой сети



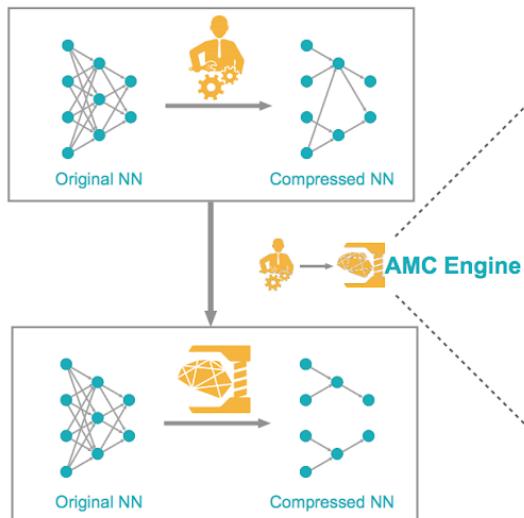
→ Последние работы в этом направлении предлагают использовать [машины методы](#) для этой задачи

→ Иначе говоря, обучать нейронную сеть, которая будет сжимать [другую нейронную сеть](#)

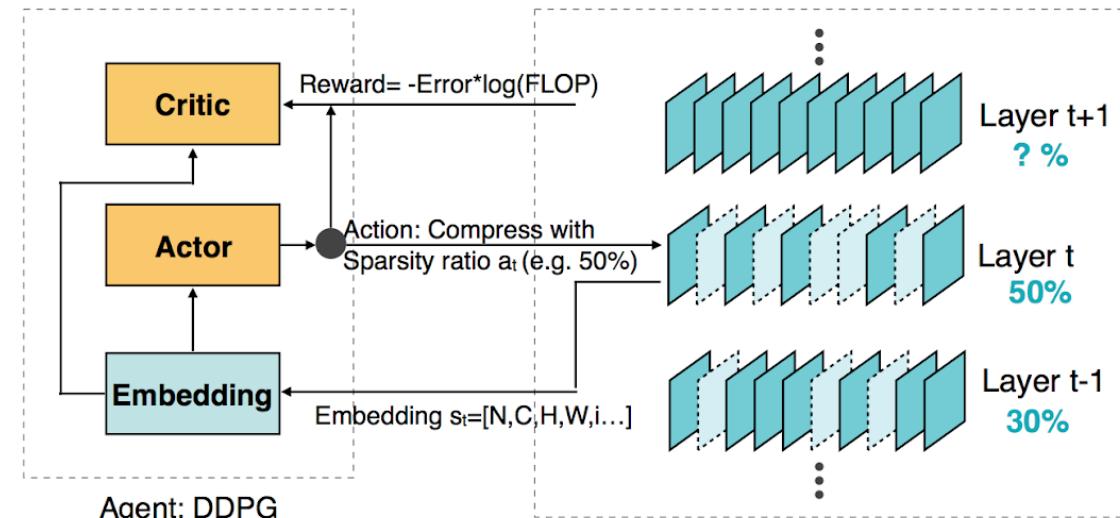


→ В 2018 году авторы представили подход «AutoML for Model Compression» (AMC), предлагающий обучать модель машинного обучения, которая бы эффективно сжимала другую сеть

Model Compression by Human:
Labor Consuming, Sub-optimal



Model Compression by AI:
Automated, Higher Compression Rate, Faster



Environment: Channel Pruning

→ Авторы указывают на то, что им удалось сжать некоторые популярные архитектуры на 70% без потери качества

Модель	Точность (%)	Время работы (мс)
MobileNetV1	70.9	123
MobileNetV1 – 50%	70.5	68.9
MobileNetV2 – 70%	70.9	-

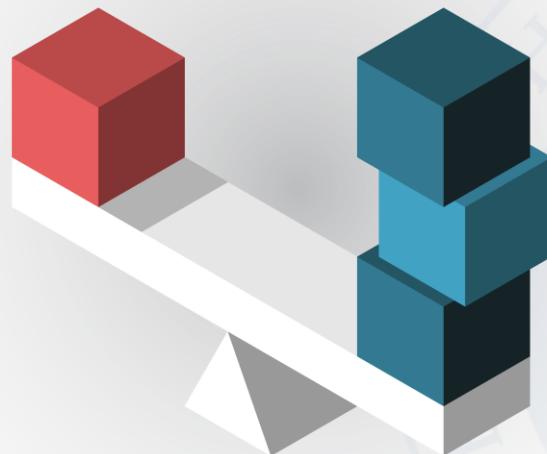
Выводы

- Нейронные сети часто имеют больше параметров, чем требуется для решения задачи



Выводы

- Нейронные сети часто имеют больше параметров, чем требуется для решения задачи
- Удаление лишних параметров — простой и достаточно эффективный способ уменьшить размер сети и ускорить ее работу

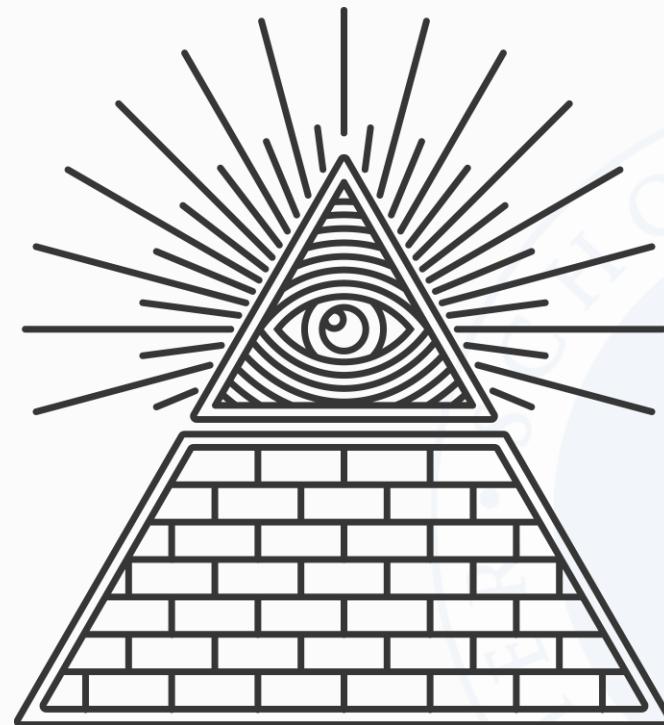


Дистилляция



Скрытые знания

→ Внутри гигантского количества параметров в большой сети скрывается много знания про устройство объектов из обучающей выборки



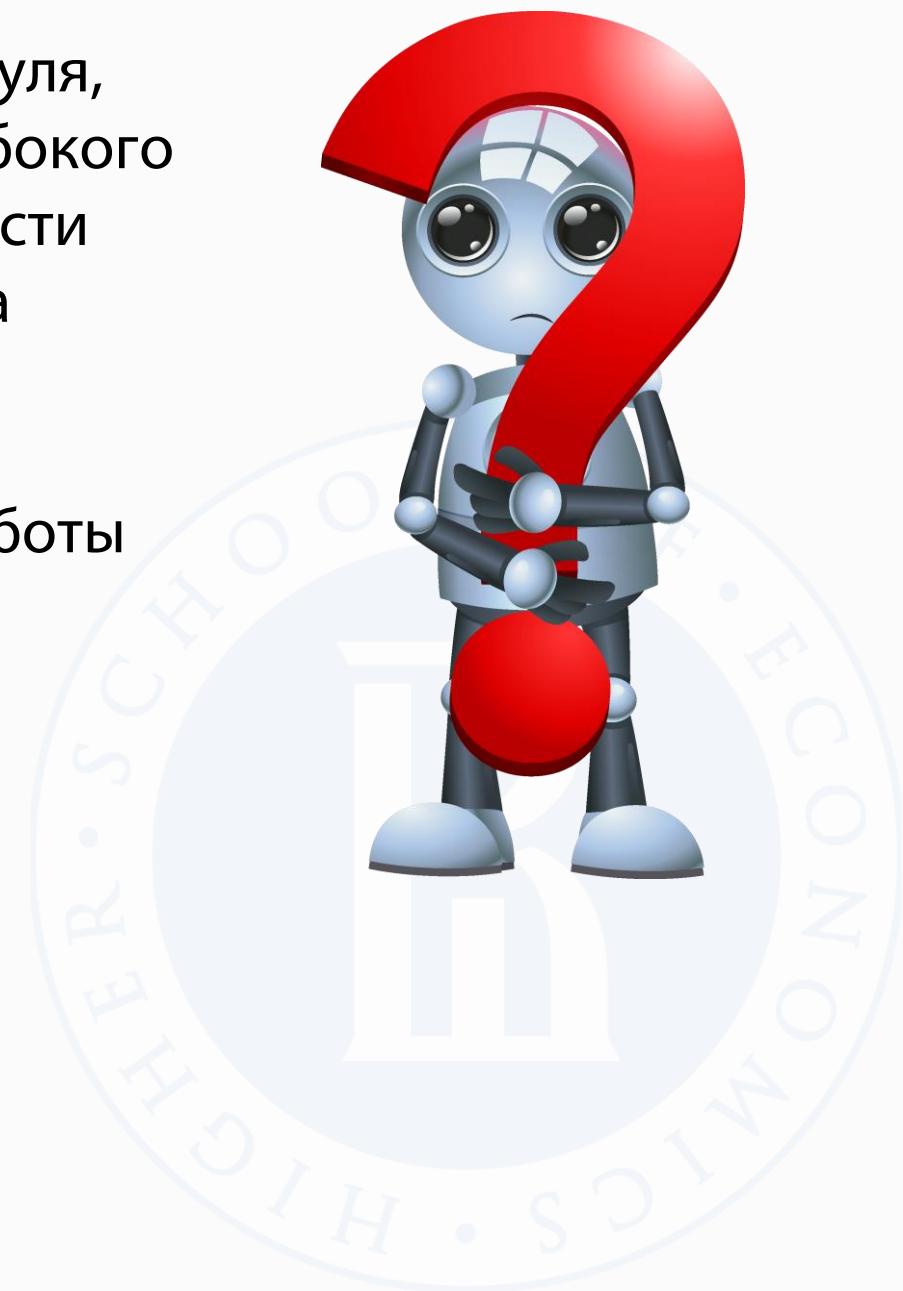
Скрытые знания

→ При работе такой сети мы их явно никак не используем.
Нам интересен лишь **итоговый ответ** сети



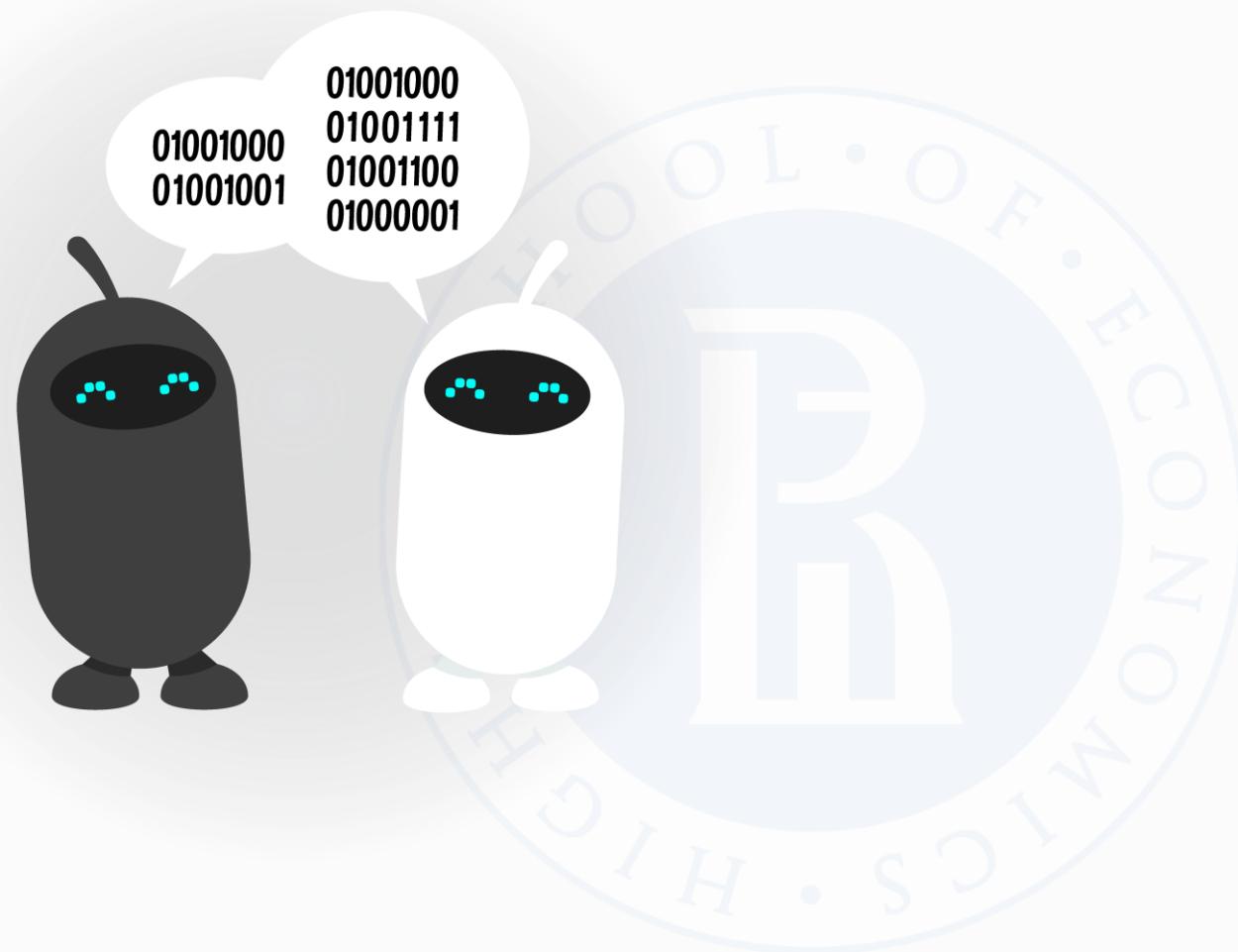
Скрытые знания

- Маленькая сеть, обучаясь с нуля, может **не обрести** такого глубокого **понимания** предметной области из-за **небольшого** количества параметров
- Соответственно, **качество** работы будет также сильно **ниже**



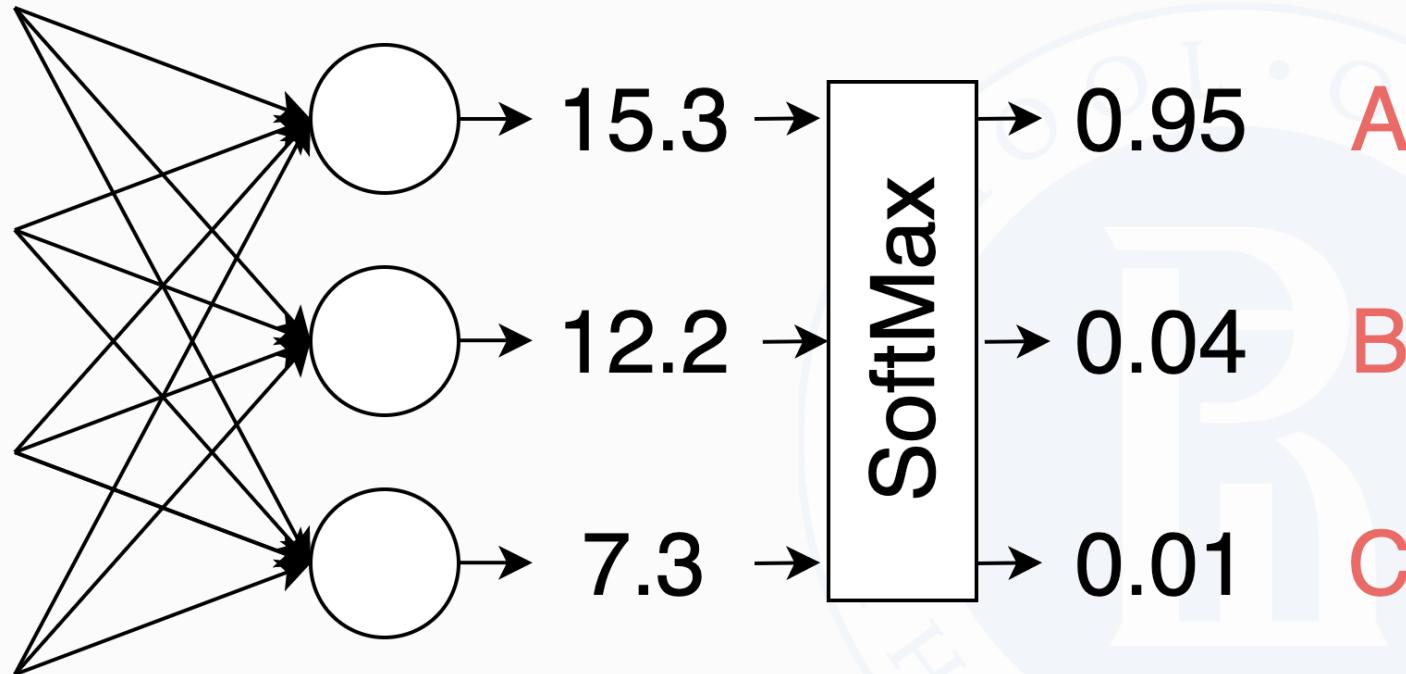
Скрытые знания

→ Возникает идея: каким-то образом от большой модели передавать эти скрытые знания маленькой, в процессе ее обучения



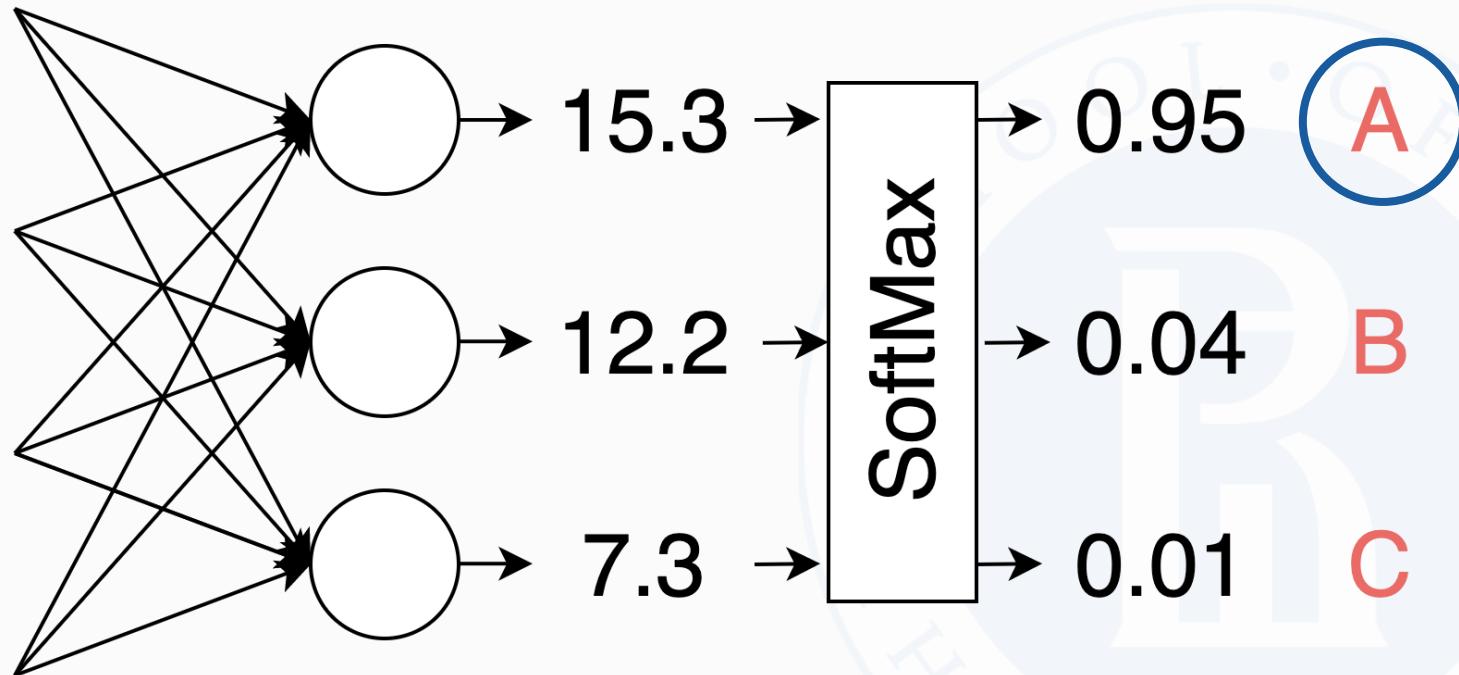
Интуиция модели

→ Давайте посмотрим на **последний** слой сети. Чаще всего там присутствует **SoftMax**, который превращает выходы последнего слоя в **вероятности классов**



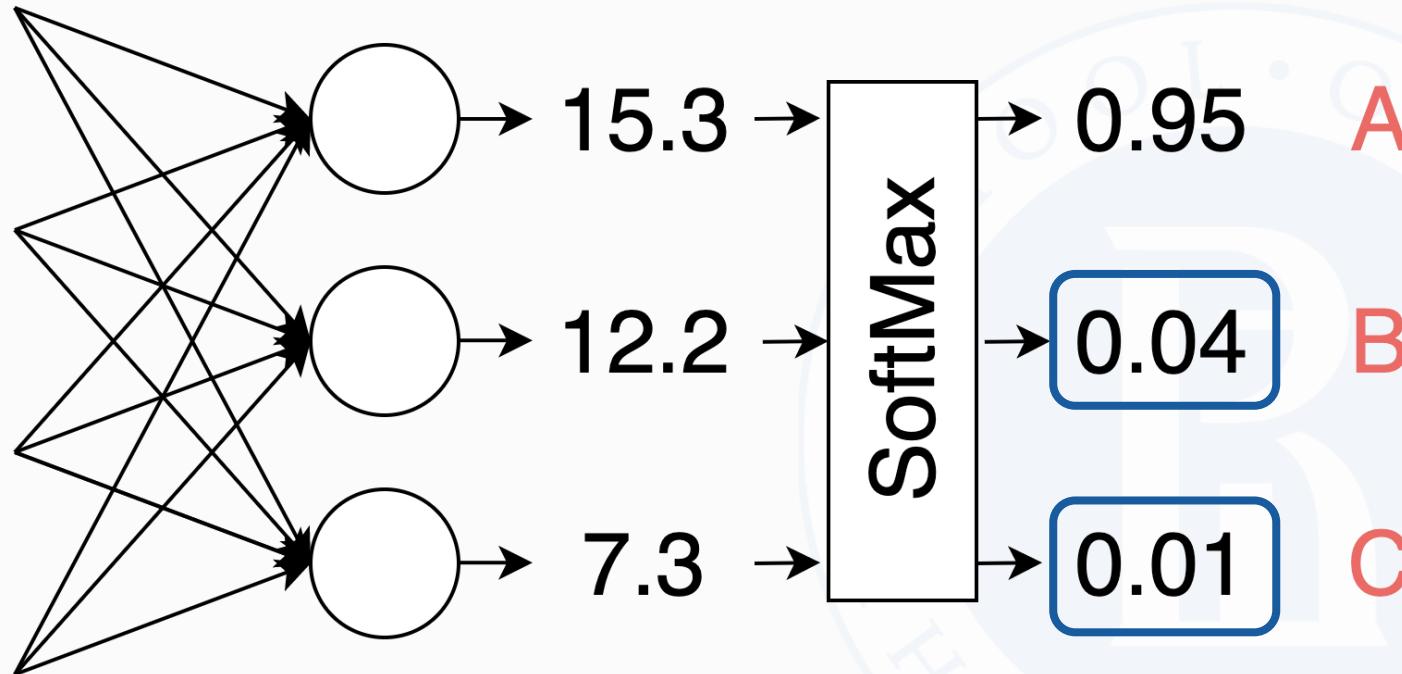
Интуиция модели

→ В этом примере сеть уверена в том, что это класс А на 95%, поэтому результатом работы сети считаем класс А



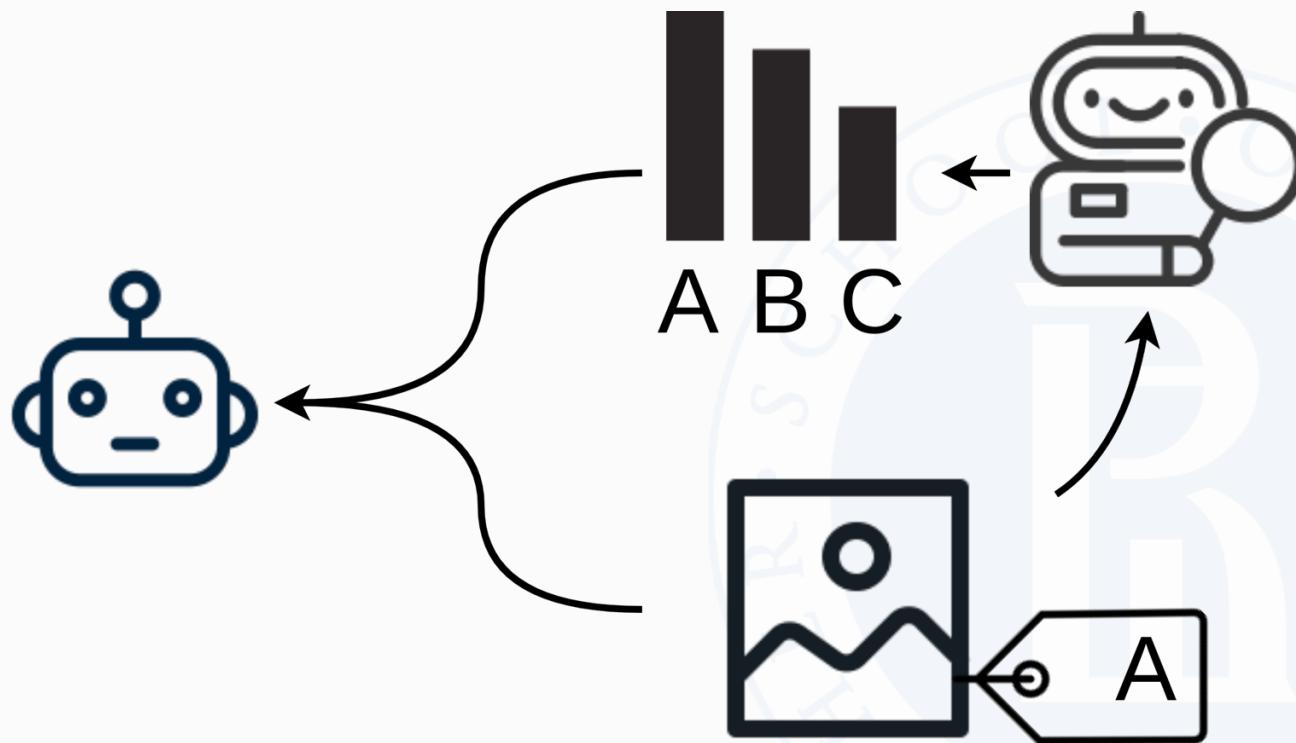
Интуиция модели

→ При этом мы видим, как сеть оценивает и **все остальные классы** на правдоподобность. Неформально можно назвать это интуицией модели



Интуиция модели

→ Можно попробовать передавать маленькой модели не только правильный ответ для объекта, но и интуицию большой модели относительно этого объекта



Интуиция модели

- Чтобы понять, как это может улучшить качество маленькой модели, рассмотрим пример
- Есть 4 размеченные картинки. Какое здесь **правило разделения** на классы?

A



A



A



B



Интуиция модели

- Зная только правильные ответы, можно предположить, что исходный критерий — форма
- Все **круглые** — A, **квадратные** — B. От **штриховки**, кажется, вообще **ничего** не зависит

A



A



A



B



Интуиция модели

→ Теперь добавим еще данные про уверенность в том, какой это именно класс

→ Изменился ли наш выбор критерия?

A



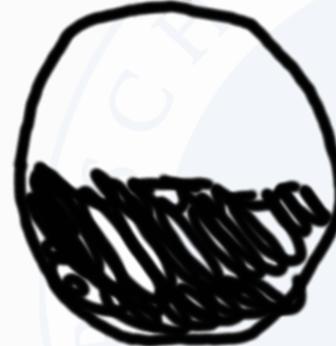
A: 99%
B: 1%

A



A: 51%
B: 49%

A



A: 52%
B: 48%

B



A: 1%
B: 99%

Интуиция модели

→ Первые три почти **одинаково** круглые, а уверенность совершенно **разная**

→ Значит, на самом деле, это **форма не имеет значения!**

A



A: 99%
B: 1%

A



A: 51%
B: 49%

A



A: 52%
B: 48%

B



A: 1%
B: 99%

Интуиция модели



А вот штриховка как раз сильно влияет:
штрихованные слева — А, штрихованные справа — В



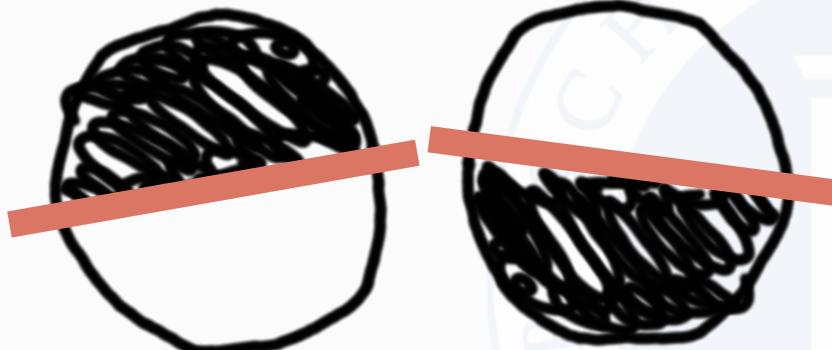
На 2 и 3 картинках можно видеть, что левая сторона
заштрихована **чуть-чуть больше**, поэтому ответ — А

A



A: 99%
B: 1%

A



A: 51%
B: 49%

A



A: 52%
B: 48%

B



A: 1%
B: 99%

Интуиция модели



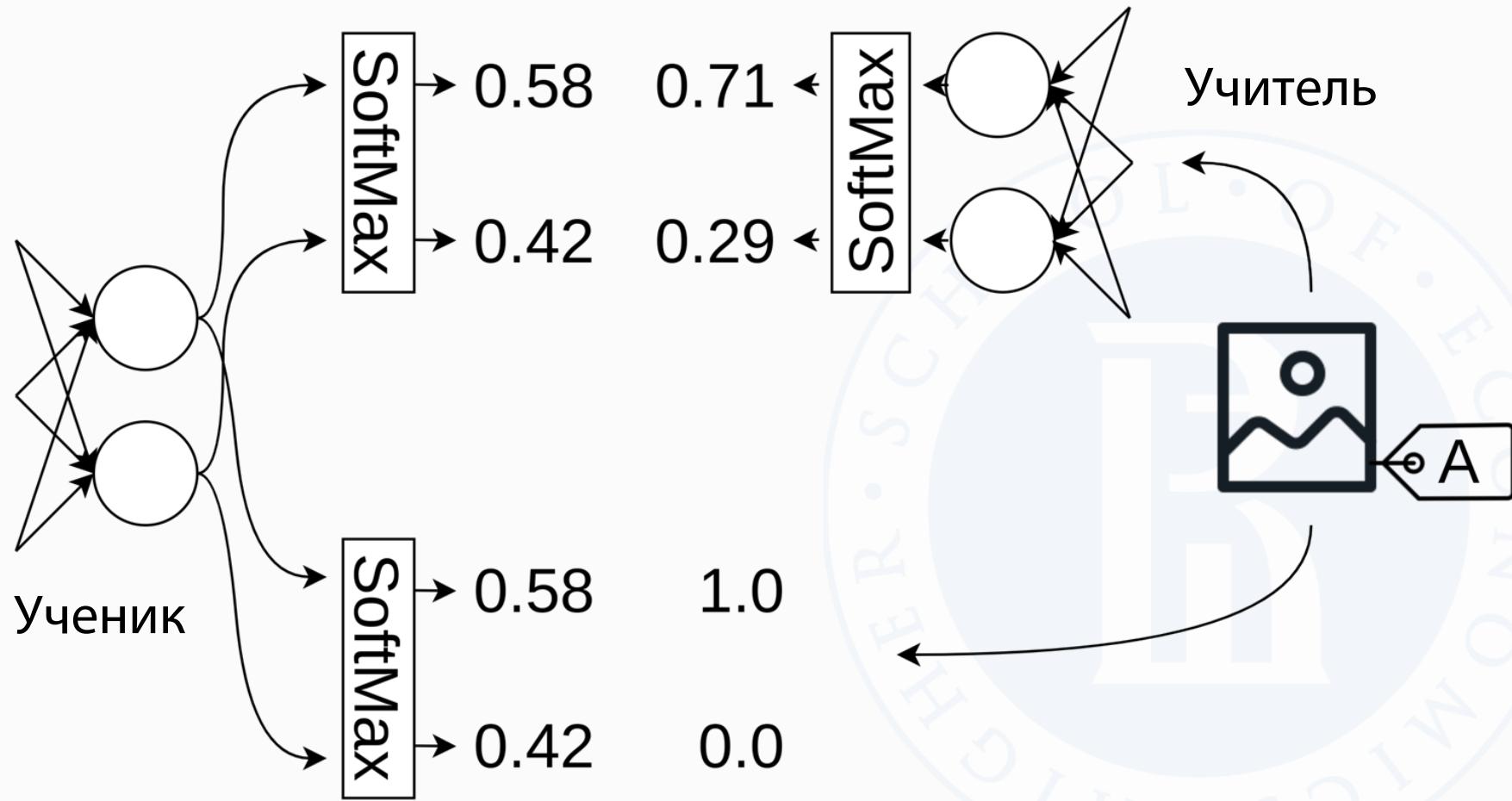
Таким образом, добавляя интуицию от большой обученной модели, мы можем подсказывать маленькой модели, на что именно нужно обращать внимание при обучении



Интуиция модели

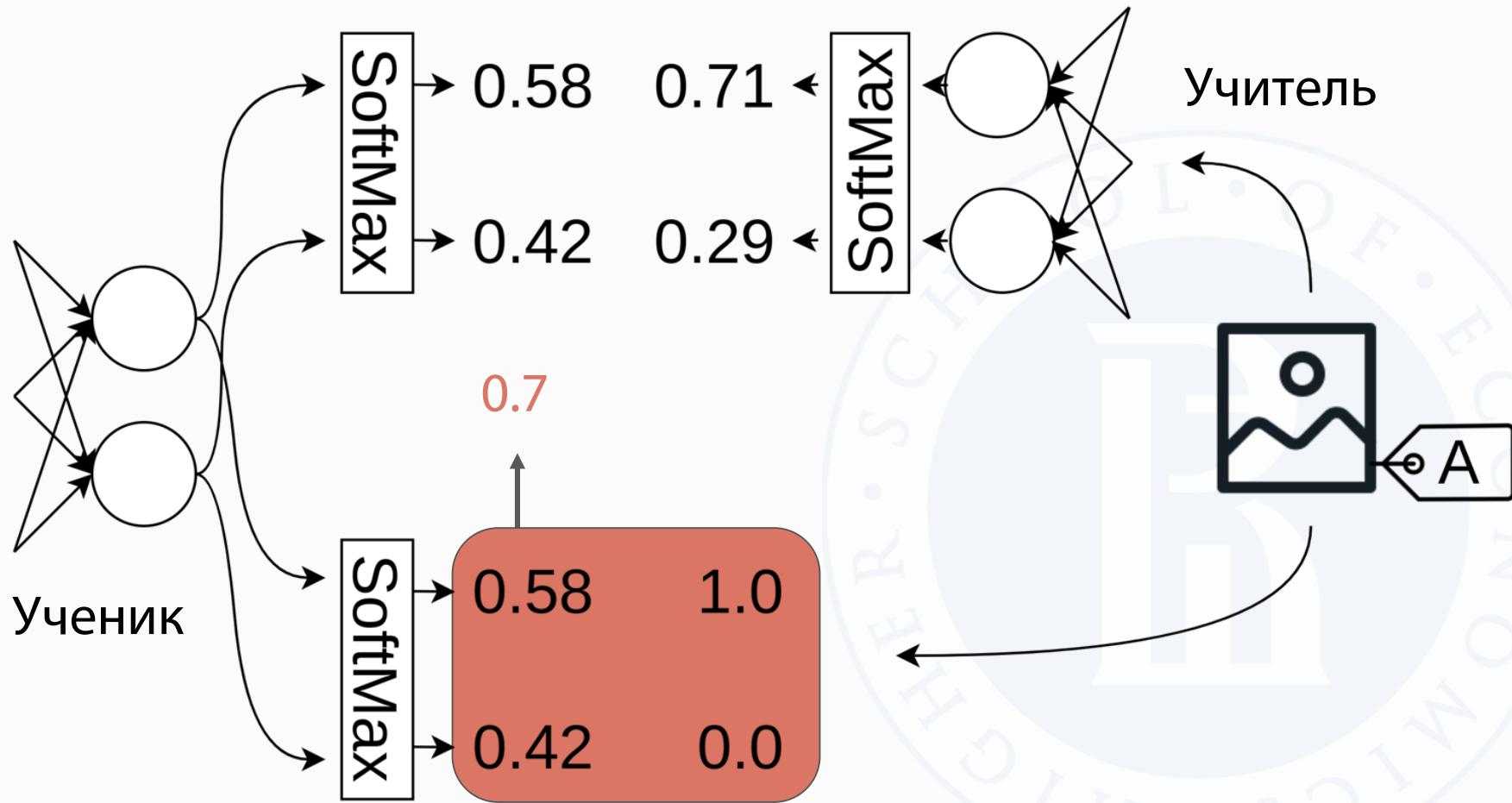


Технически мы просто делаем более сложную функцию ошибки



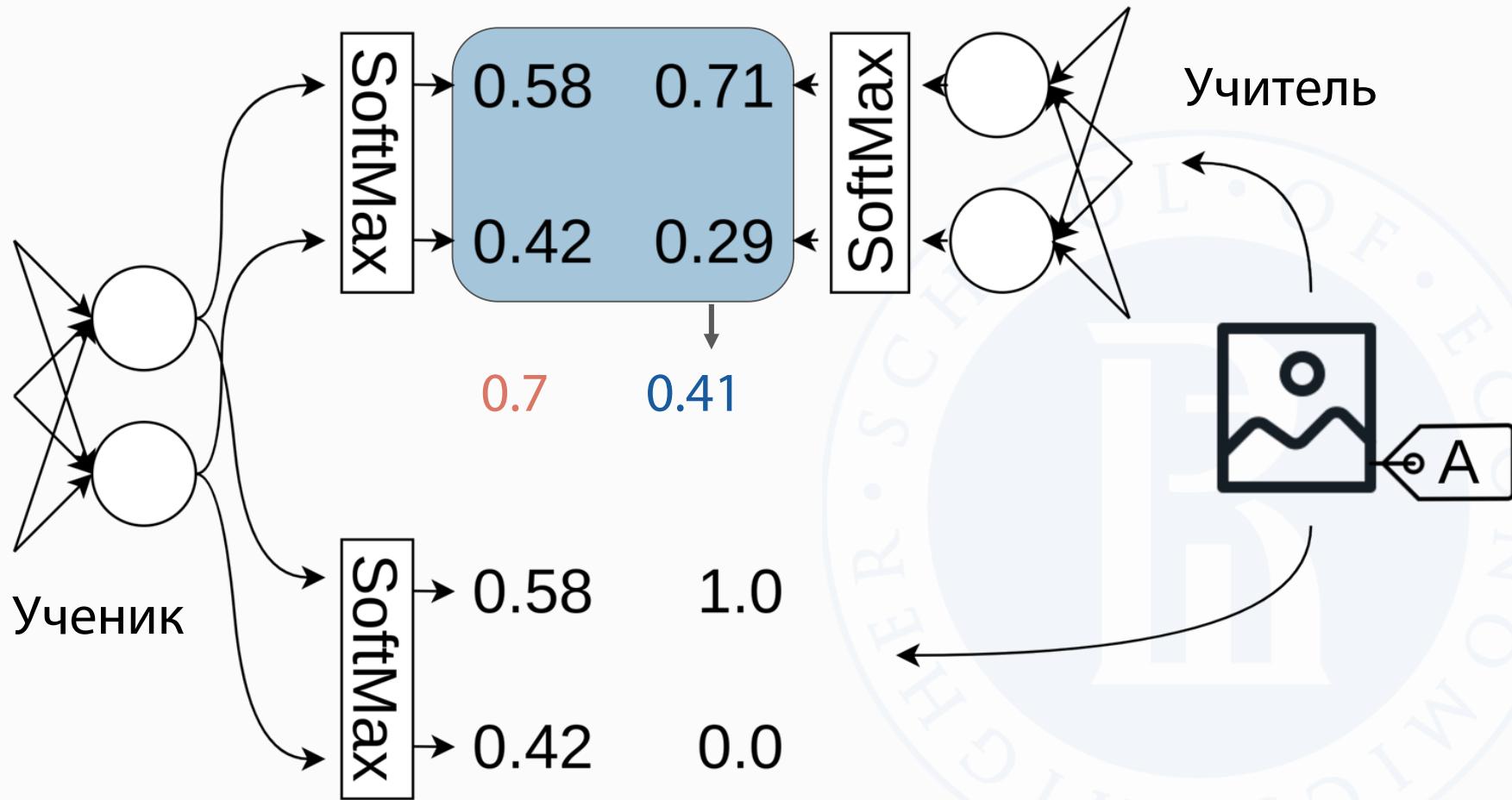
Интуиция модели

→ Как обычно считаем ошибку, сравнивая через кросс-энтропию с **правильным ответом**



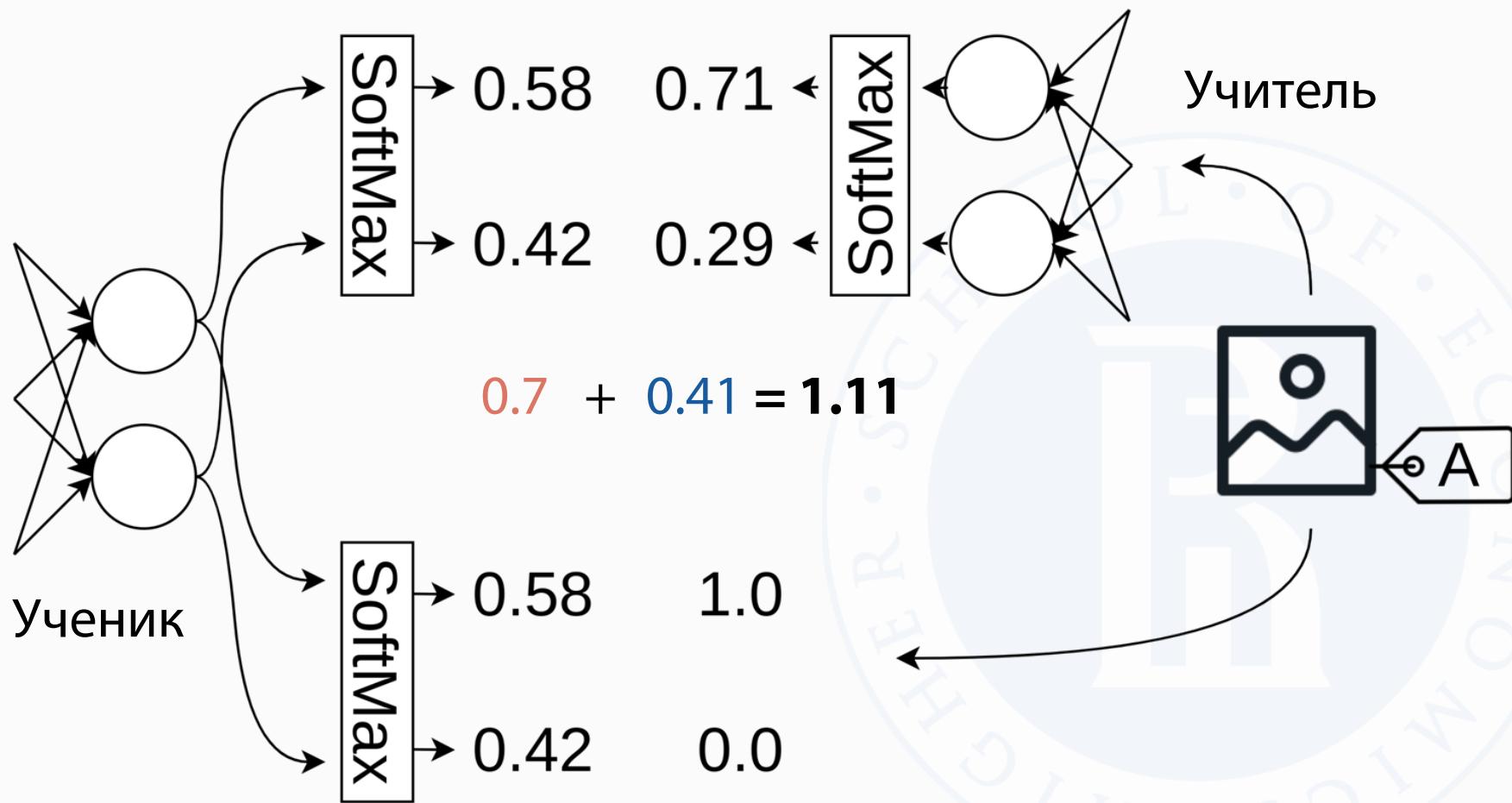
Интуиция модели

→ Считаем вероятности большой модели и через кросс-энтропию сравниваем с ответом маленькой модели



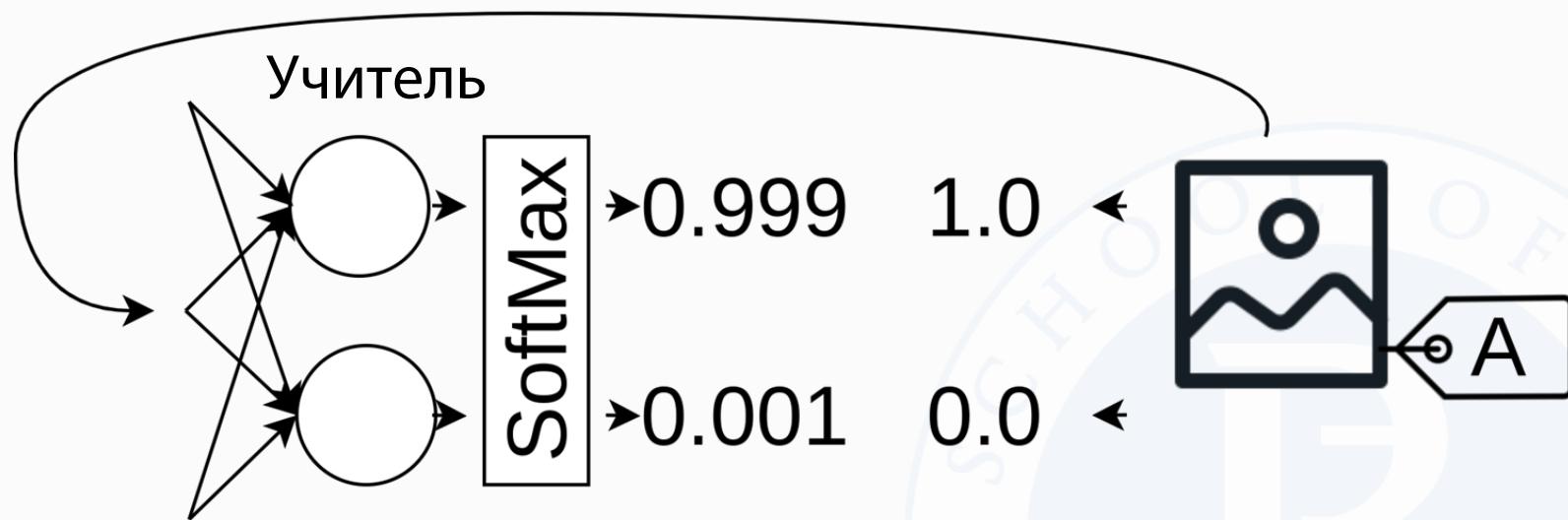
Интуиция модели

→ Складываем — это и будет функцией ошибки, которую нужно будет минимизировать



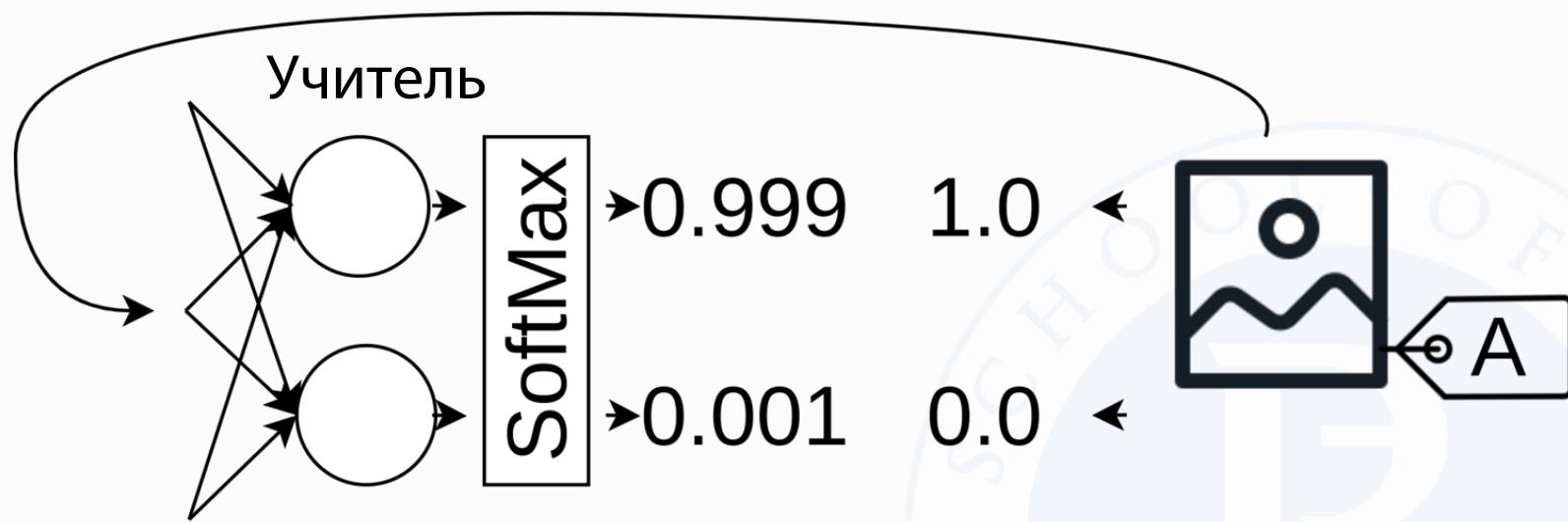
Нагретый SoftMax

→ Если большая модель очень хорошо обучена, то может возникнуть проблема — она **слишком уверена** в своих ответах



Нагретый SoftMax

- Если большая модель очень хорошо обучена, то может возникнуть проблема — она **слишком уверена** в своих ответах



- В таком случае, ее предсказания будут **практически совпадать с правильным ответом**, и маленькая модель почти **никакой новой информации** не получит

Нагретый SoftMax

→ Чтобы исправить эту проблему, используют SoftMax с дополнительным параметром T (температура)

Нагретый SoftMax

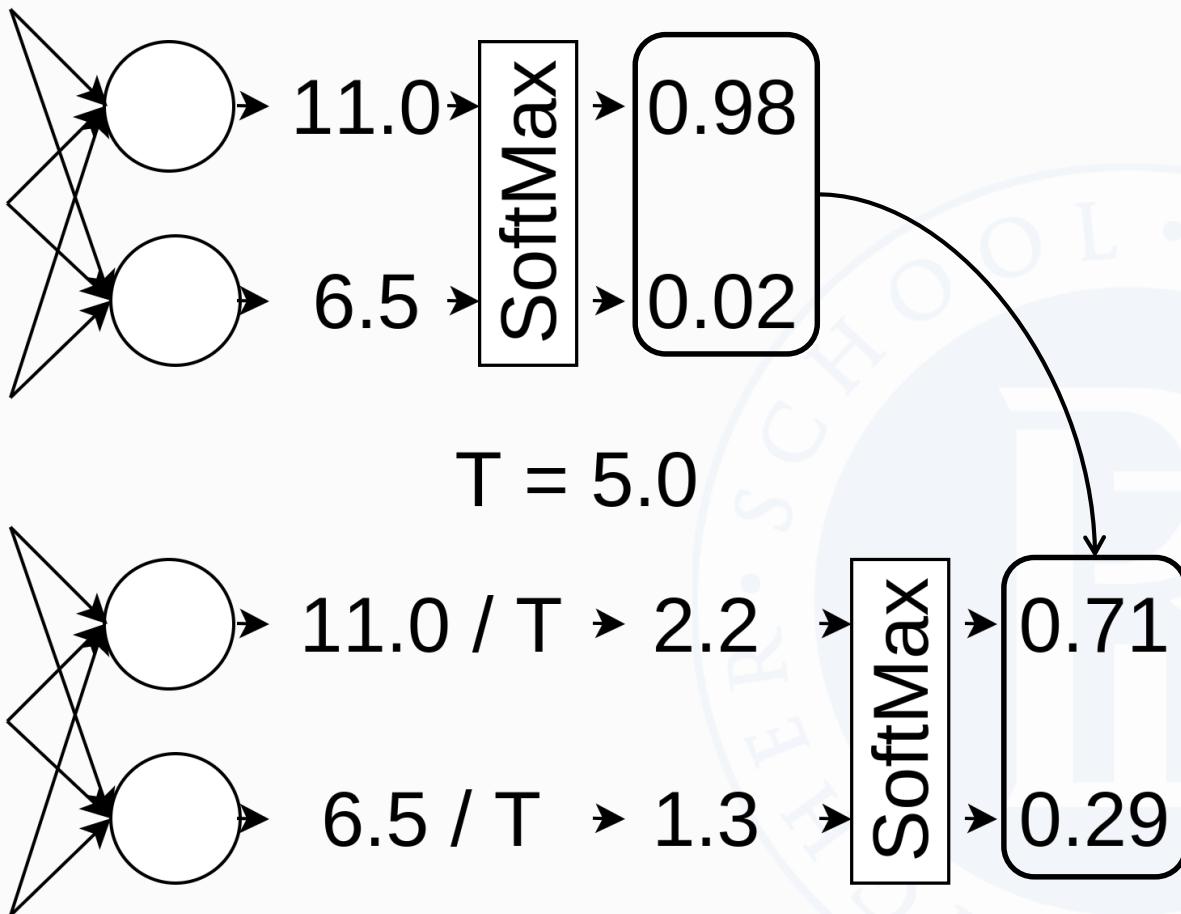
- Чтобы исправить эту проблему, используют SoftMax с дополнительным параметром T (температура)
- Все выходы сети перед SoftMax просто [делятся на число \$T\$](#)

$$HSoftMax(T) = \text{SoftMax}(X / T)$$

Нагретый SoftMax



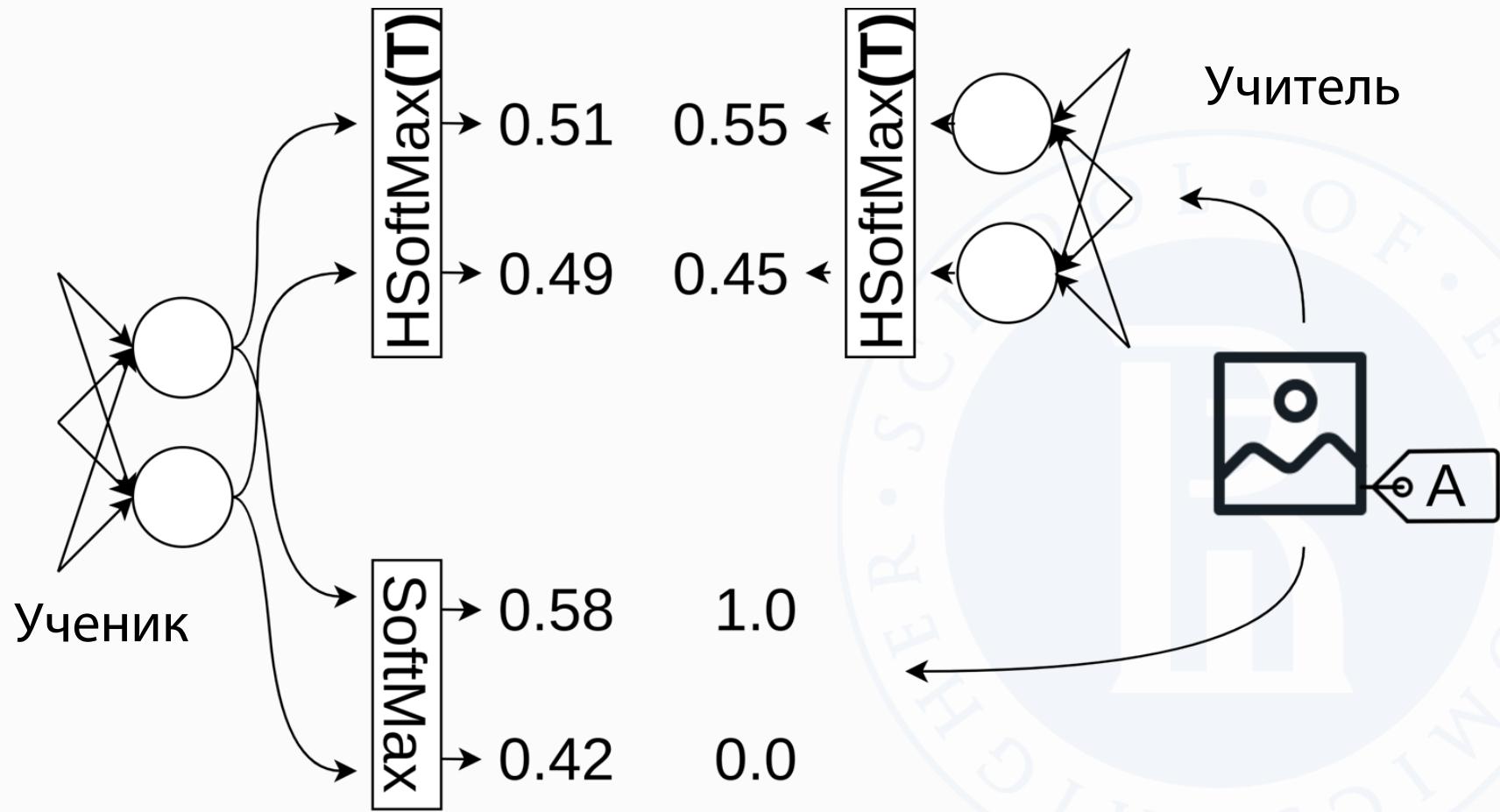
Это позволяет «смягчить» ответы сети и получить **больше информации** про распределение ответов



Нагретый SoftMax

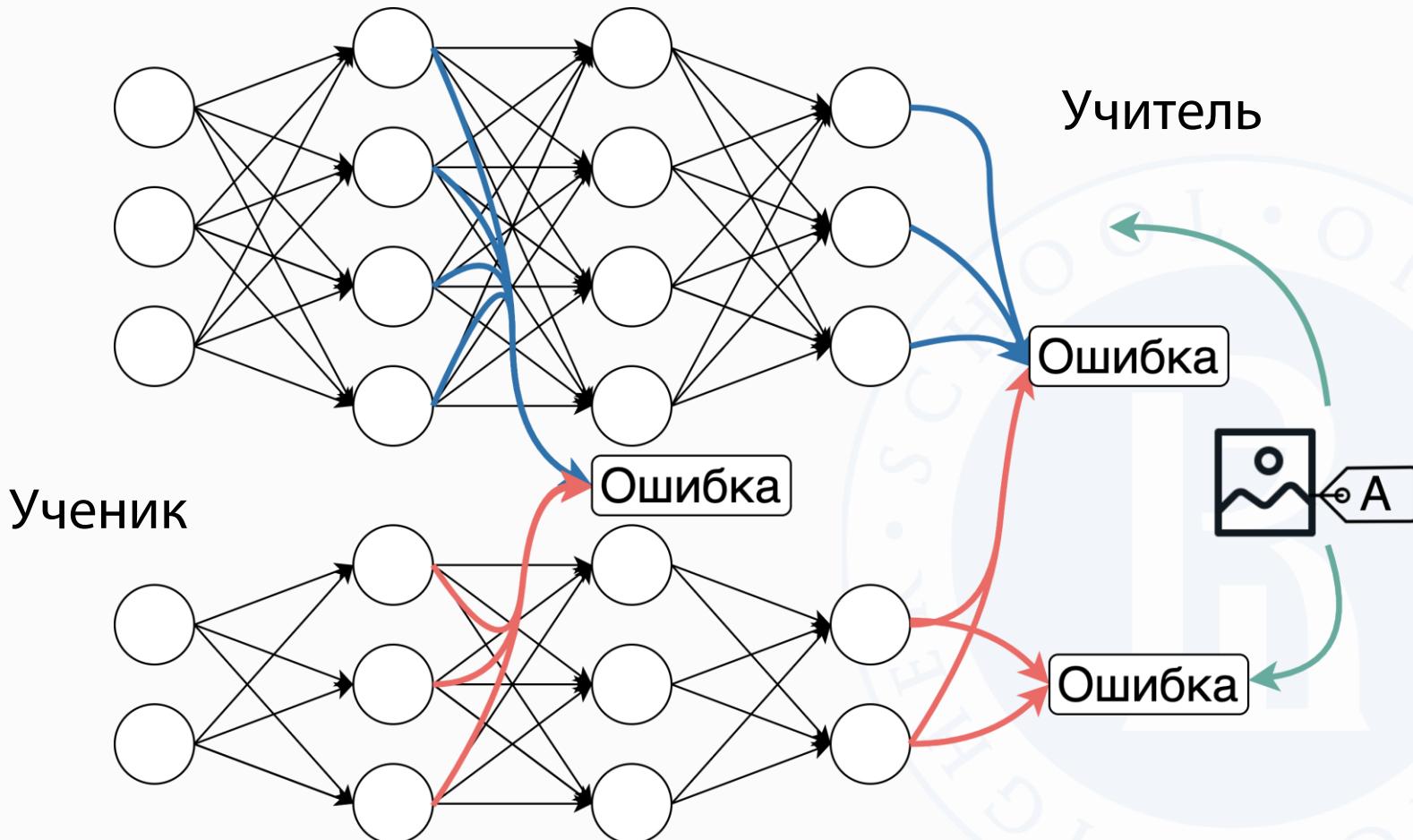


Важно отметить, что «нагретый» SoftMax нужно использовать **и для сети-учителя, и для сети-ученика**



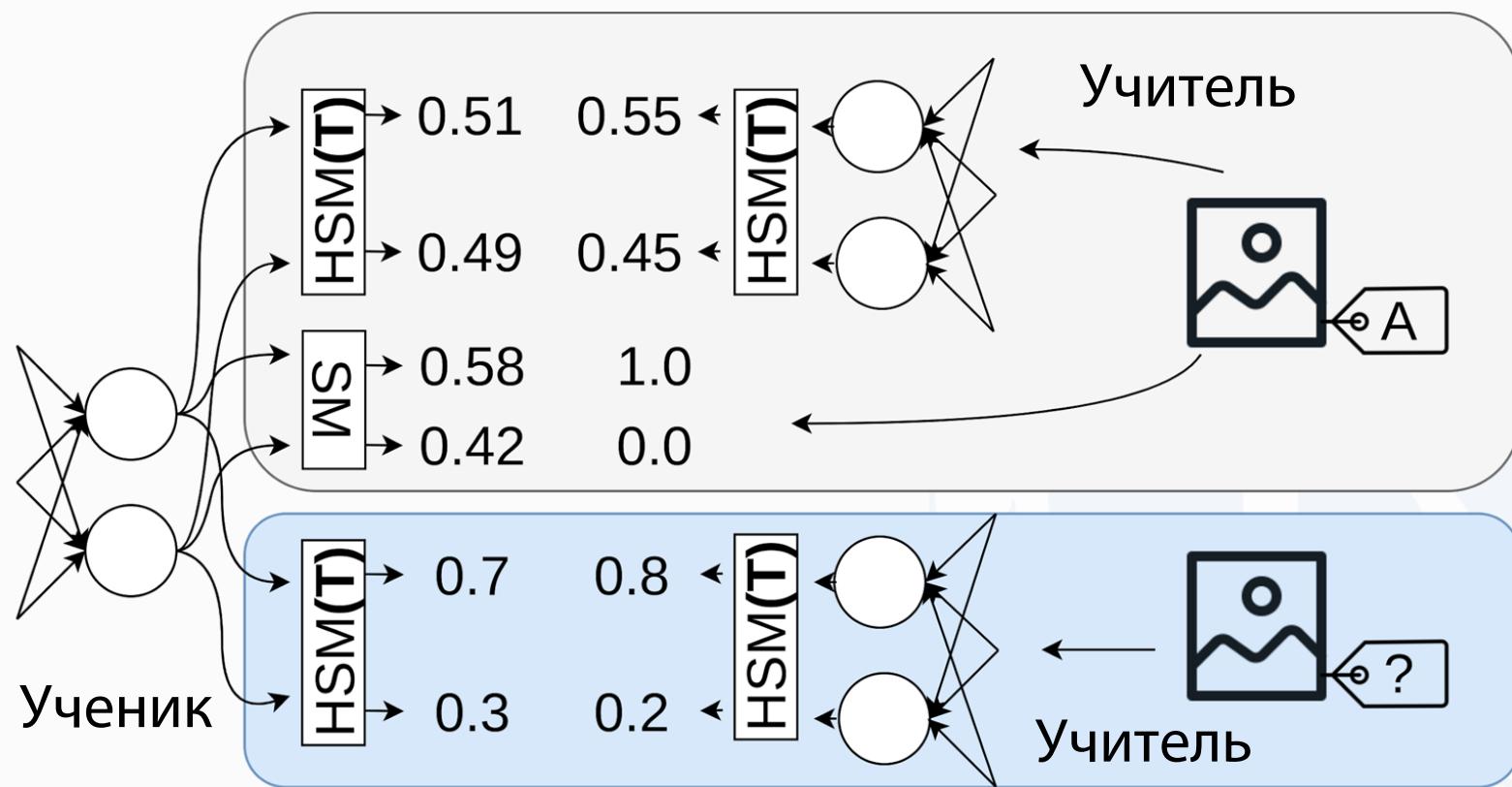
Более глубокие знания

→ Если сети имеют **схожую структуру**, то можно сравнивать не только последние, но и **более глубокие** слои



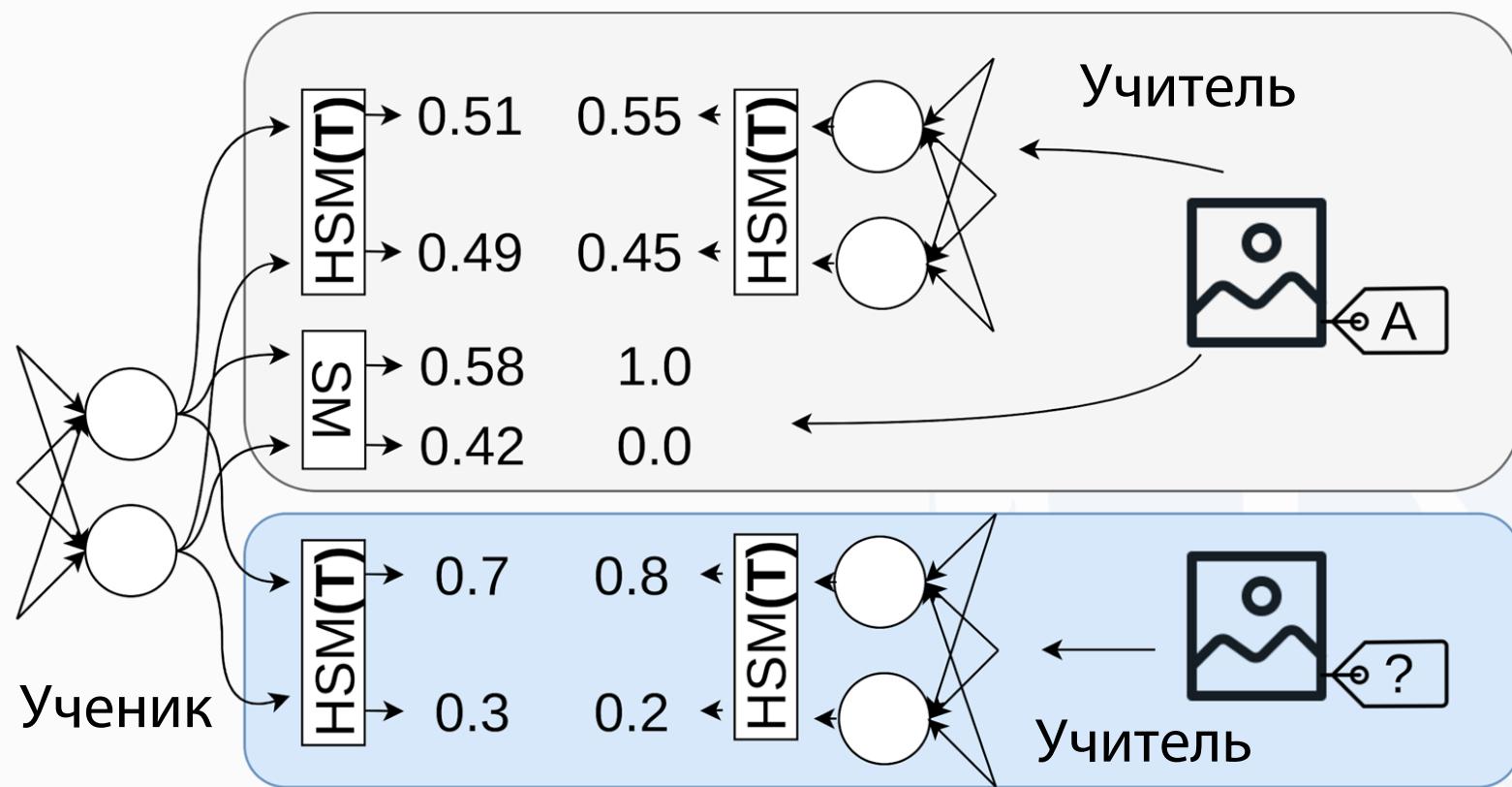
Обучение на неразмеченных данных

→ Так как сеть-учитель делает **качественные** предсказания, то можно попробовать с помощью нее **доразметить новые данные** и добавить их в обучение сети-ученика



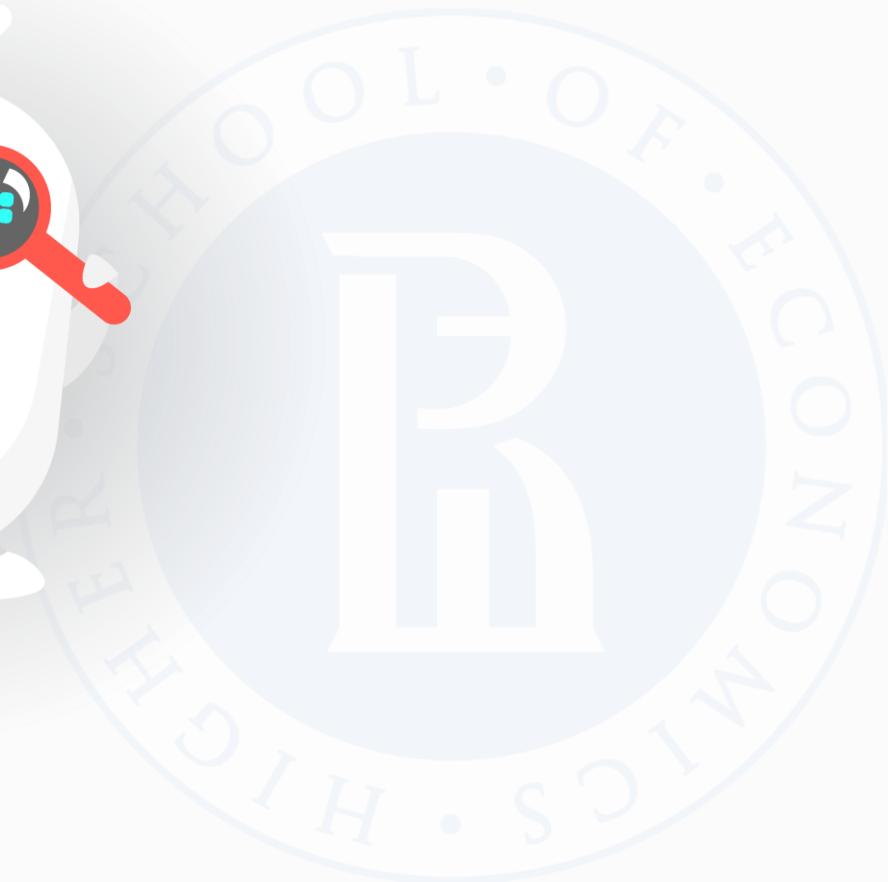
Обучение на неразмеченных данных

→ Так как ответы сети все же **не идеальны**, лучше использовать **«нагретый SoftMax»** для извлечения интуиции модели



Итог

- Используя дистилляцию, можно улучшать качество обучения небольших моделей, добавляя им **дополнительную информацию** про предметную область



Квантизация



Операции на процессоре

→ Числа в разном формате по-разному представляются внутри компьютера:

INT8



1 Байт (знак + число)

INT32



4 Б (знак + число)

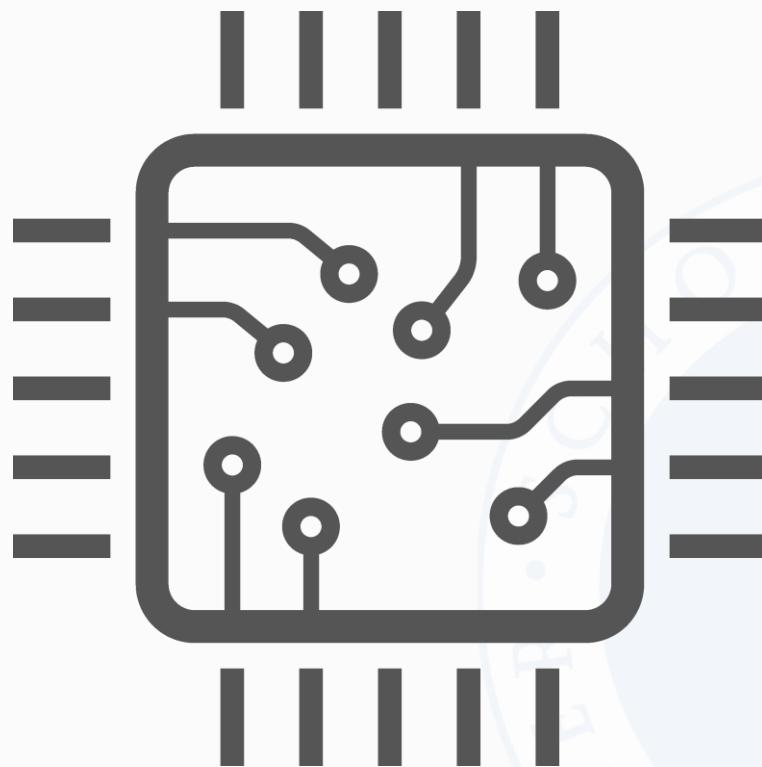
FLOAT32



4 Б (знак + экспонента + мантисса)

Операции на процессоре

→ Операции для разных представлений чисел на процессоре стоят по-разному



Операции на процессоре

→ Ресурсоемкость операций над разными типами

	Память	Такты процессора
INT8	2 байта	1-3 такта
FLOAT32	8 байт	2-8 тактов

→ Реальные числа зависят от типа процессора, однако общий итог такой:

Операции на процессоре



Ресурсоемкость операций над разными типами

	Память	Такты процессора
INT8	2 байта	1-3 такта
FLOAT32	8 байт	2-8 тактов

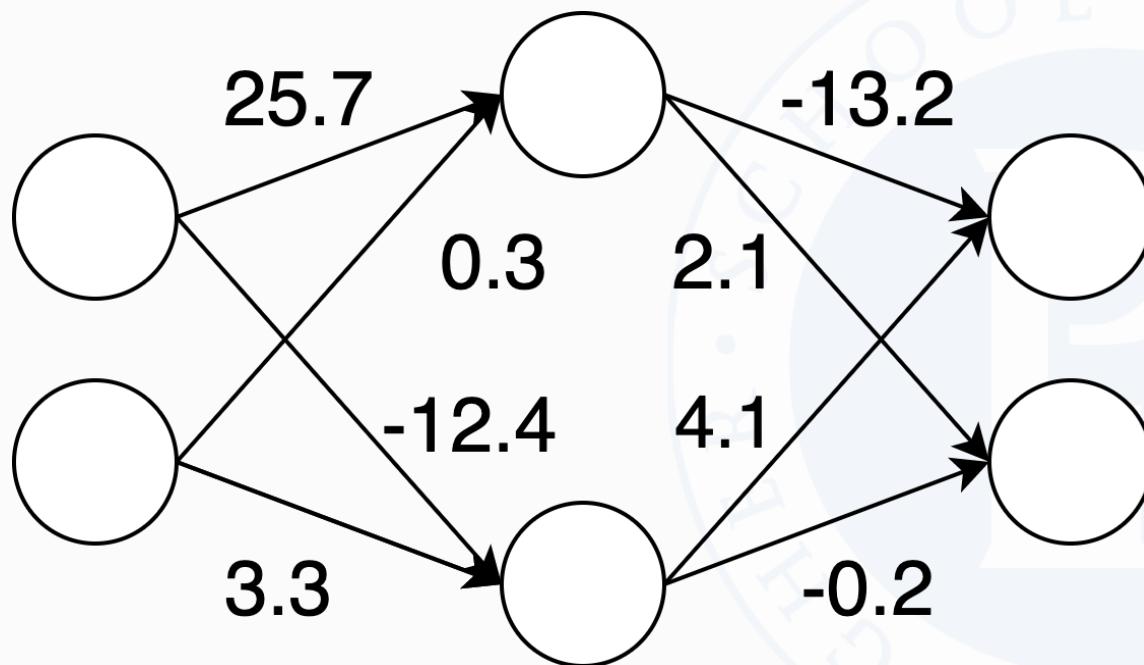


Реальные числа зависят от типа процессора, однако общий итог такой:

- Операции над **FLOAT32** требуют **больше памяти и времени** для вычисления, чем **INT8**

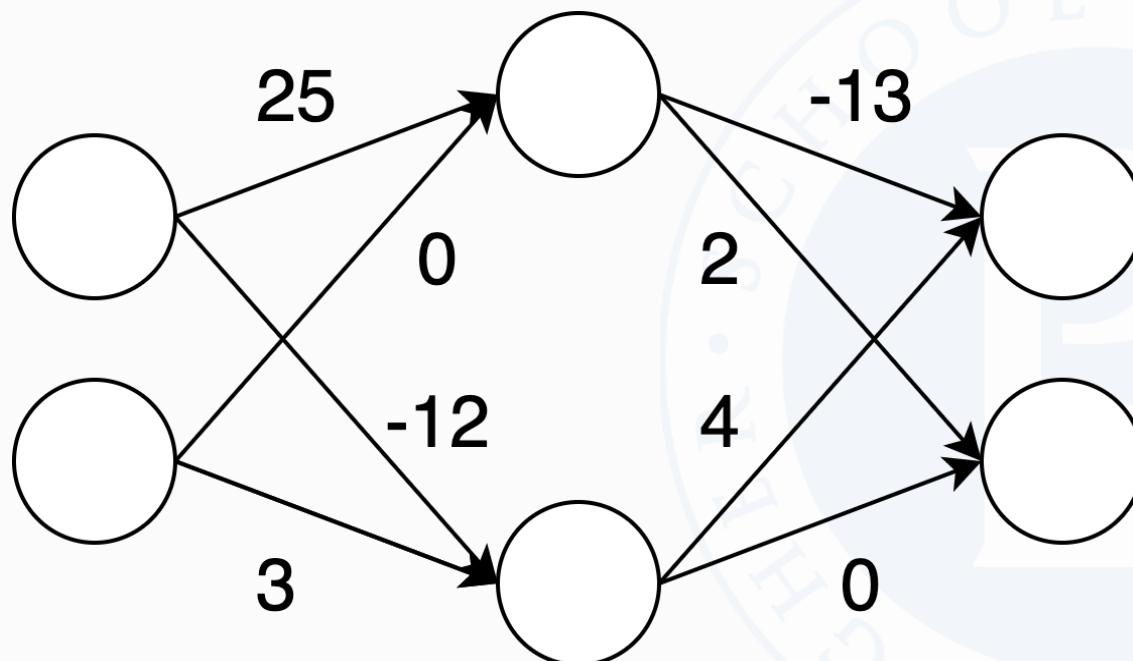
Параметры сети

→ Обычно все параметры сети — это дробные числа, записанные в формате FLOAT32



Параметры сети

- Возникает мысль — если «загрузить» сеть и заменить FLOAT32 на INT8, то сеть будет работать быстрее и занимать в 4 раза меньше места
- Но останется ли качество приемлемым?



Квантование чисел



INT8 умеет хранить числа от -128 до 127 включительно.
Поэтому если мы просто округлим все числа, у нас могут
возникнуть следующие проблемы:

Квантование чисел

- INT8 умеет хранить числа от -128 до 127 включительно.
Поэтому если мы просто округлим все числа, у нас могут возникнуть следующие проблемы:
- Слишком маленький диапазон чисел
- Если все веса **близки к одному числу**, то мы можем **потерять** слишком много информации

Квантование чисел

- INT8 умеет хранить числа от -128 до 127 включительно.
Поэтому если мы просто округлим все числа, у нас могут возникнуть следующие проблемы:
 - Слишком маленький диапазон чисел
 - Если все веса **близки к одному числу**, то мы можем **потерять слишком много информации**
 - Например, если наши веса **0.23, 0.78, -0.56, -0.99**, то мы их заменим на **0, 0, 0, 0**.
Что, очевидно, плохо — мы **потеряли всю информацию**

Квантование чисел



INT8 умеет хранить числа от -128 до 127 включительно.
Поэтому если мы просто округлим все числа, у нас могут
возникнуть следующие проблемы:

Квантование чисел

- INT8 умеет хранить числа от -128 до 127 включительно.
Поэтому если мы просто округлим все числа, у нас могут возникнуть следующие проблемы:
 - Слишком большой диапазон чисел
 - Если изначальные числа слишком большие, то, округляя до ближайшего, также теряем много информации

Квантование чисел

- INT8 умеет хранить числа от -128 до 127 включительно.
Поэтому если мы просто округлим все числа, у нас могут возникнуть следующие проблемы:
 - Слишком большой диапазон чисел
 - Если изначальные числа слишком большие, то, округляя до ближайшего, также теряем много информации
 - Например, если наши веса
742.32, 156.45, -256.89, -1237.89
То мы их заменим на
127, 127, -128, -128
Что также плохо — информации почти не осталось

Квантование чисел

→ Таким образом, необходимо более **аккуратно** переводить изначальный диапазон в [-128; 127], стараясь терять **минимум информации**

Квантование чисел

- Таким образом, необходимо более аккуратно переводить изначальный диапазон в [-128; 127], стараясь терять минимум информации
- Для этого будем дополнительно растягивать и смещать числа — формула для этого следующая

$$X_{\text{INT8}} = \text{round}(X_{\text{FLOAT}} / S + Z)$$

Квантование чисел

- Таким образом, необходимо более аккуратно переводить изначальный диапазон в [-128; 127], стараясь терять минимум информации
- Для этого будем дополнительно растягивать и смещать числа — формула для этого следующая

$$X_{\text{INT8}} = \text{round}(X_{\text{FLOAT}} / S + Z)$$

- Где S отвечает за растяжение, а Z — за смещение

Квантование чисел

- Таким образом, необходимо более аккуратно переводить изначальный диапазон в [-128; 127], стараясь терять минимум информации
- Для этого будем дополнительно растягивать и смещать числа — формула для этого следующая

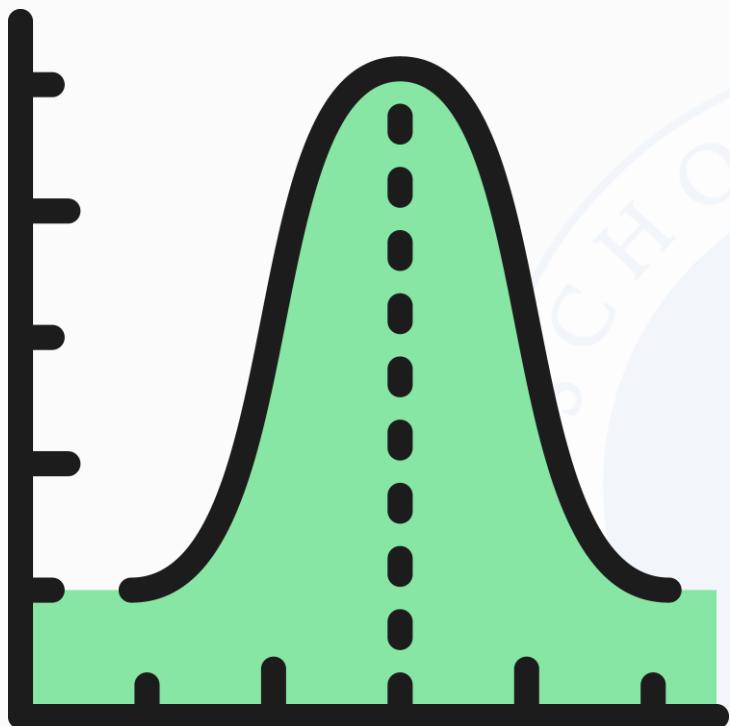
$$X_{\text{INT8}} = \text{round}(X_{\text{FLOAT}} / S + Z)$$

- Где S отвечает за растяжение, а Z — за смещение
- Формула восстановления исходного числа очень проста

$$X_{\text{FLOAT}} = (X_{\text{INT}} - Z) \cdot S$$

Квантование чисел

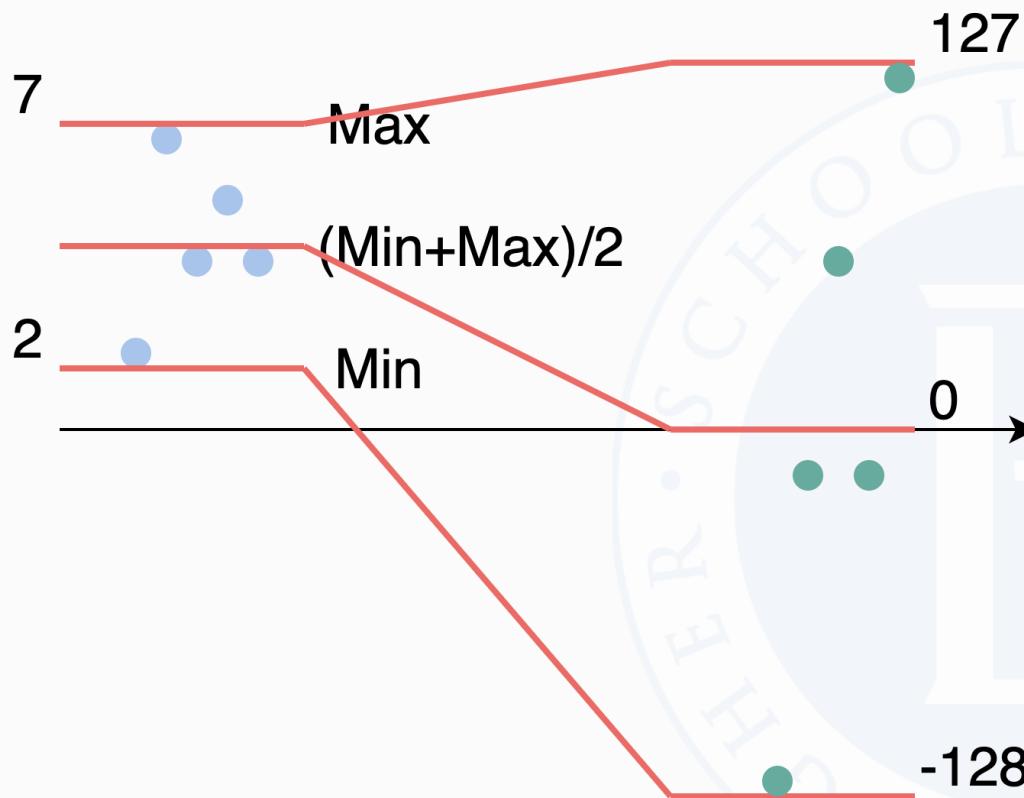
→ Сами параметры смещения можно подбирать, смотря на [распределение исходных данных](#). Наиболее простой и популярный метод — [MinMax](#)



Квантование чисел

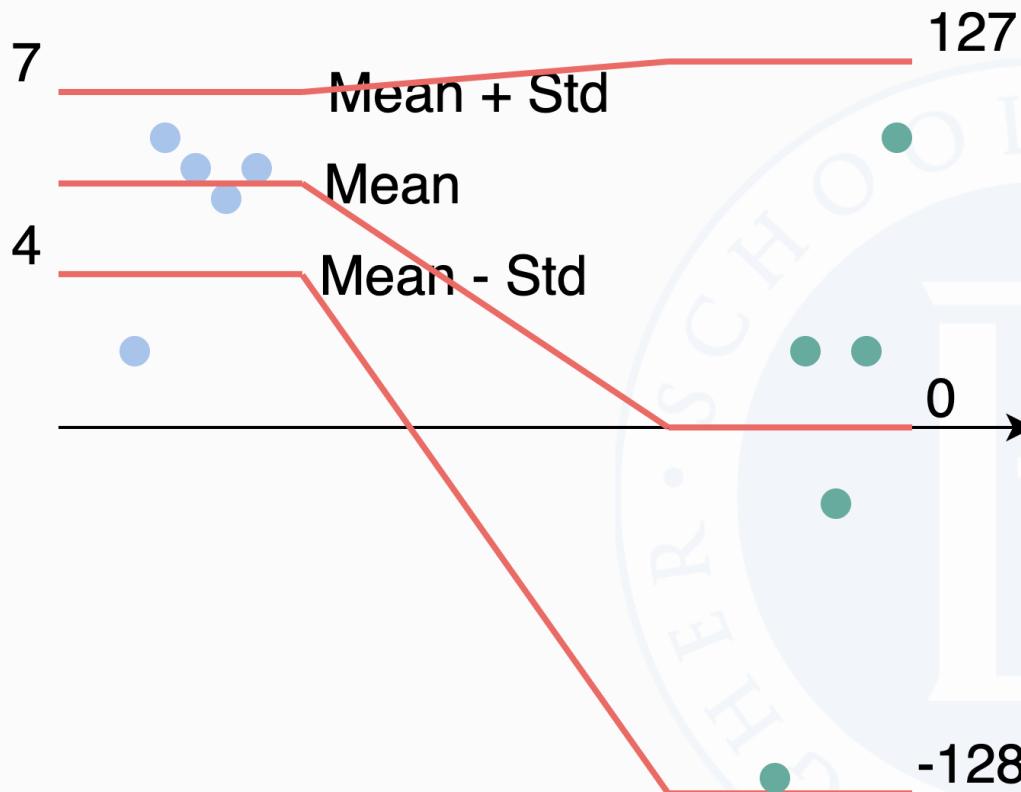
$$S = [\text{Max}(X) - \text{Min}(X)] / 255$$

$$Z = [\text{Max}(X) + \text{Min}(X)] / (-2 \cdot S)$$



Квантование чисел

→ Стоит отметить, что [существуют и другие методы](#), которые могут использовать среднее, дисперсию, медиану, квантили и другие статистические характеристики



Квантование чисел

→ Для иллюстрации, квантуем те числа, которые были в нашем примере, через MinMax

Квантование чисел

→ Для иллюстрации, квантуем те числа, которые были в нашем примере, через MinMax

$$X_{\text{FLOAT}} = 0.23, 0.78, -0.56, -0.99$$

$$S = (0.78 - (-0.99)) / 255 = 0.00694$$

$$Z = (0.78 - 0.99) / (-2 \cdot 0.00694) = 15.12711$$

$$X_{\text{INT}} = 48, 127, -65, -128$$

Квантование чисел

→ Для иллюстрации, квантуем те числа, которые были в нашем примере, через MinMax

$$X_{\text{FLOAT}} = 0.23, 0.78, -0.56, -0.99$$

$$S = (0.78 - (-0.99)) / 255 = 0.00694$$

$$Z = (0.78 - 0.99) / (-2 \cdot 0.00694) = 15.12711$$

$$X_{\text{INT}} = 48, 127, -65, -128$$

→ Теперь видно, что мы потеряли гораздо меньше информации про веса

Квантование чисел

→ Однако какая-то информация **все-таки потерялась**.
Если попробовать восстановить оригинальные значения,
то мы получим:

$$X_{\text{INT}} = 48, 127, -65, -128$$

$$X_{\text{FLOAT}} = 0.228, 0.776, -0.556, -0.993$$

Квантование чисел

→ Однако какая-то информация **все-таки потерялась**.
Если попробовать восстановить оригинальные значения,
то мы получим:

$$X_{\text{INT}} = 48, 127, -65, -128$$

$$X_{\text{FLOAT}} = 0.228, 0.776, -0.556, -0.993$$

- Видно, что значения немного отличаются от изначальных
 $0.23, 0.78, -0.56, -0.99$
- Потеря информации — **неизбежный эффект квантизации**

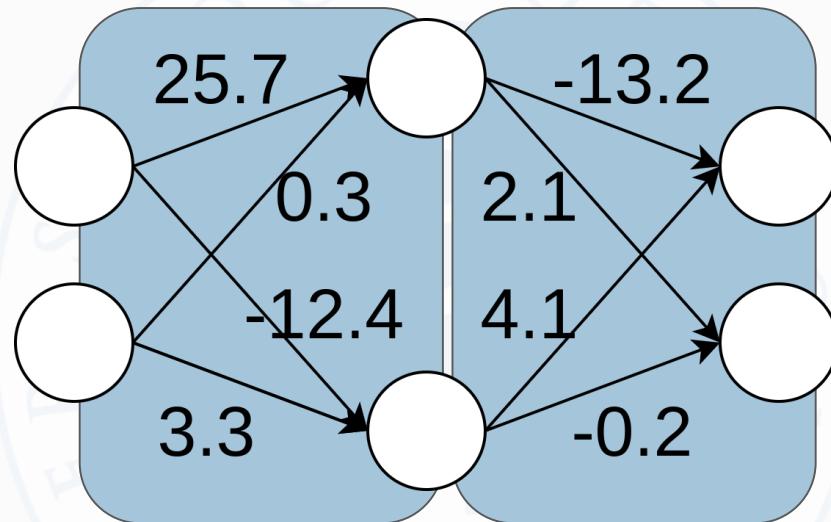
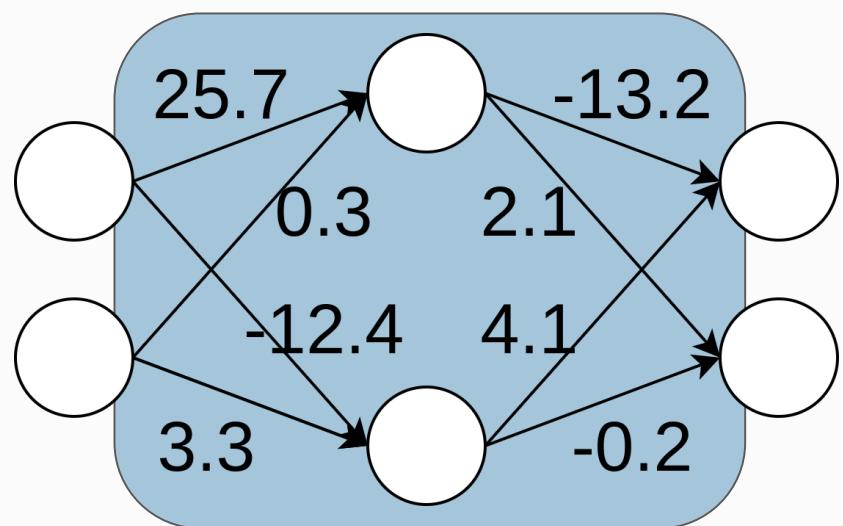
Квантование чисел

→ Важно отметить, что операции сложения и умножения можно делать **прямо в квантованном виде**. Необходимо лишь пересчитывать **новые S и Z** для нового распределения

$$A_{\text{INT}}[S_1, Z_1] \cdot B_{\text{INT}}[S_2, Z_2] = C_{\text{INT}}[S_3, Z_3]$$

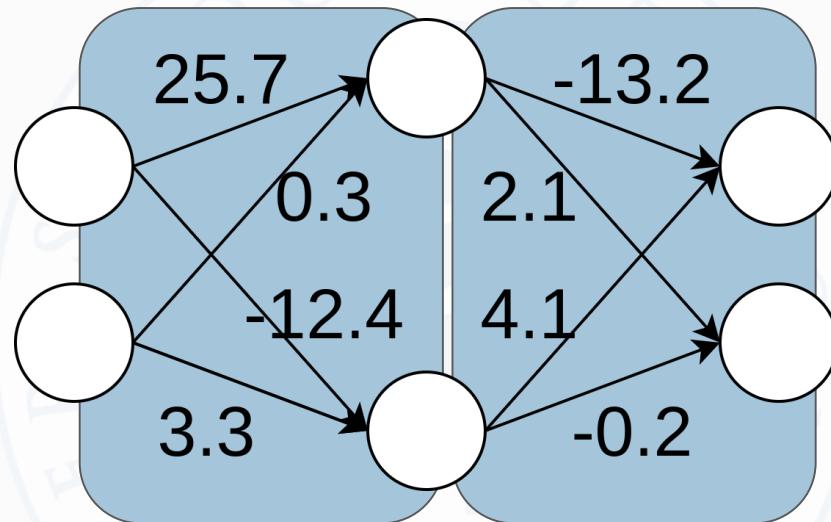
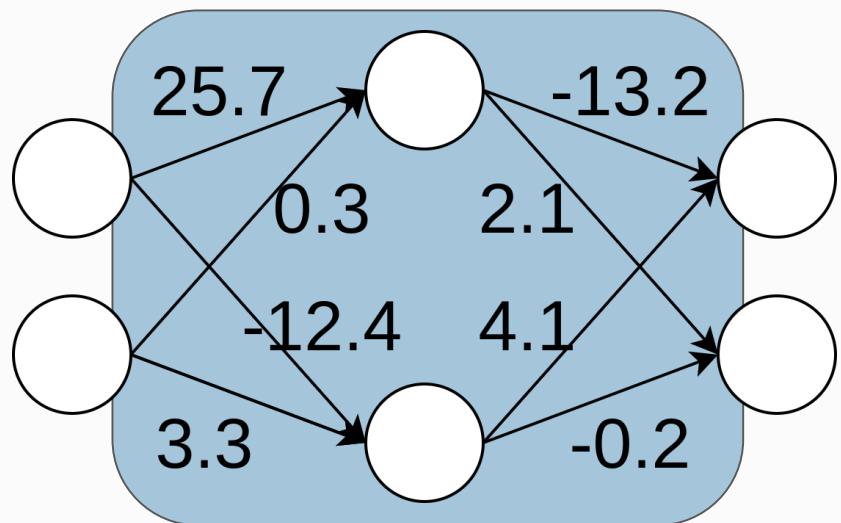
Квантование в сетях

→ Для эффективного квантования весов в сети необходимо подсчитать S и Z . Это можно делать для всей сети целиком, а можно отдельно по слоям



Квантование в сетях

- Для эффективного квантования весов в сети необходимо подсчитать S и Z . Это можно делать для всей сети целиком, а можно отдельно по слоям
- Веса сети меняться не будут — их можно фиксировать



Квантование в сетях

- С активациями нужно работать немного аккуратнее
- Так как функции активации бывают **сложными**, для эффективного вычисления они заменяются на **грубые аналоги**

Квантование в сетях

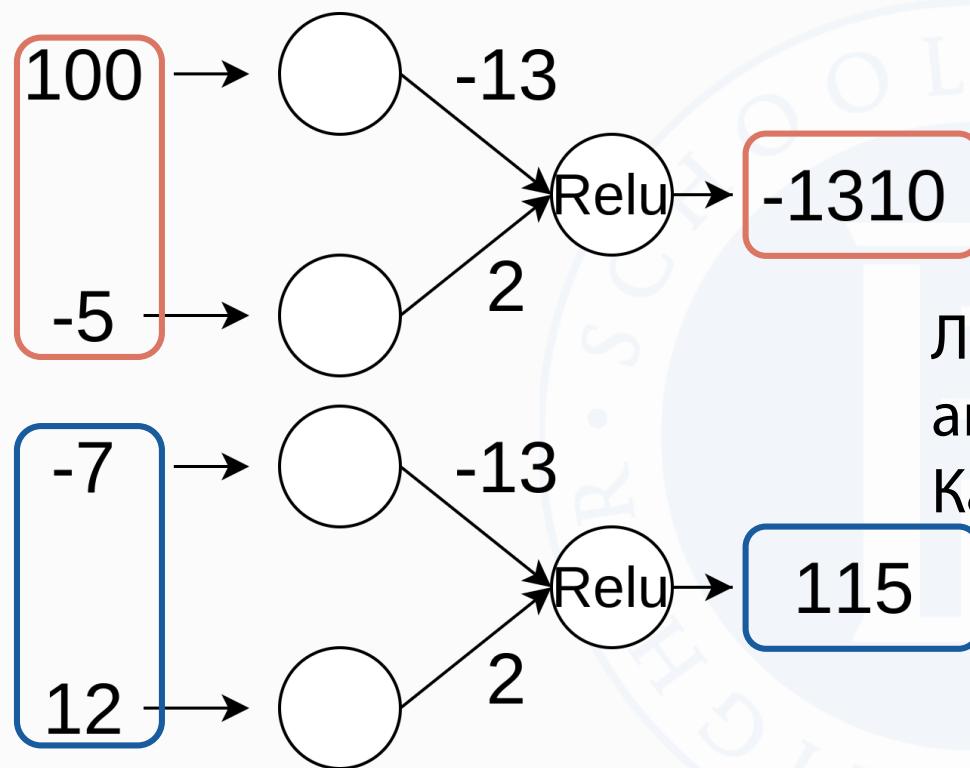
- С активациями нужно работать немного аккуратнее
- Так как функции активации бывают **сложными**, для эффективного вычисления они заменяются на **грубые аналоги**

Оригинальная активация	Сгрубленный аналог
$\sigma(x) = \frac{1}{1 + \exp(-x)}$	$\sigma_{hard}(x) = \begin{cases} 0 & x \leq -3 \\ 1 & x \geq 3 \\ x/6 + 1/2 & \text{otherwise} \end{cases}$
$Tanh(x) = \tanh(x)$	$Tanh_{hard}(x) = \begin{cases} -1 & x < -1 \\ 1 & x > 1 \\ x & \text{otherwise} \end{cases}$

Квантование в сетях

→ У активаций есть и другая проблема — их значение может быть **произвольным**, в зависимости от данных на входе.
Параметры для их квантования **неочевидны**

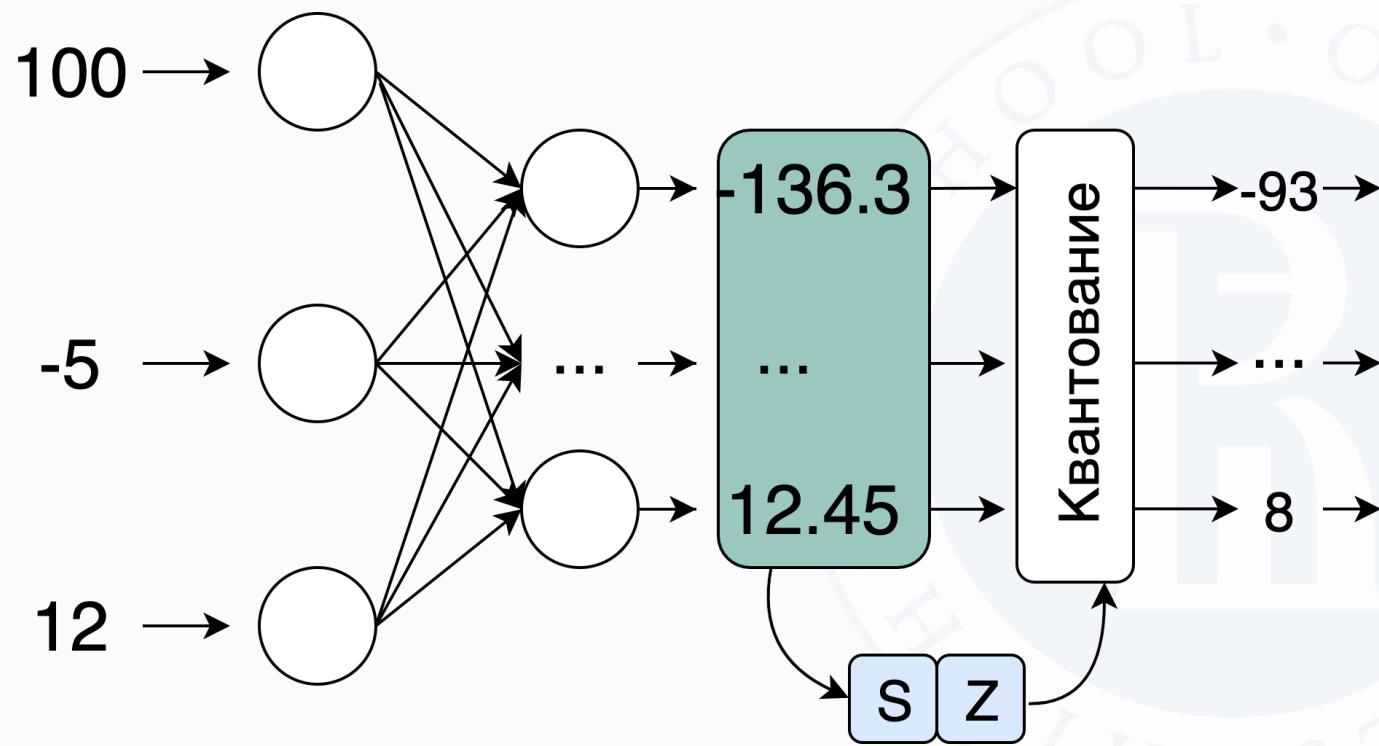
Любые
входные
данные



Любые
активации.
Как квантовать?

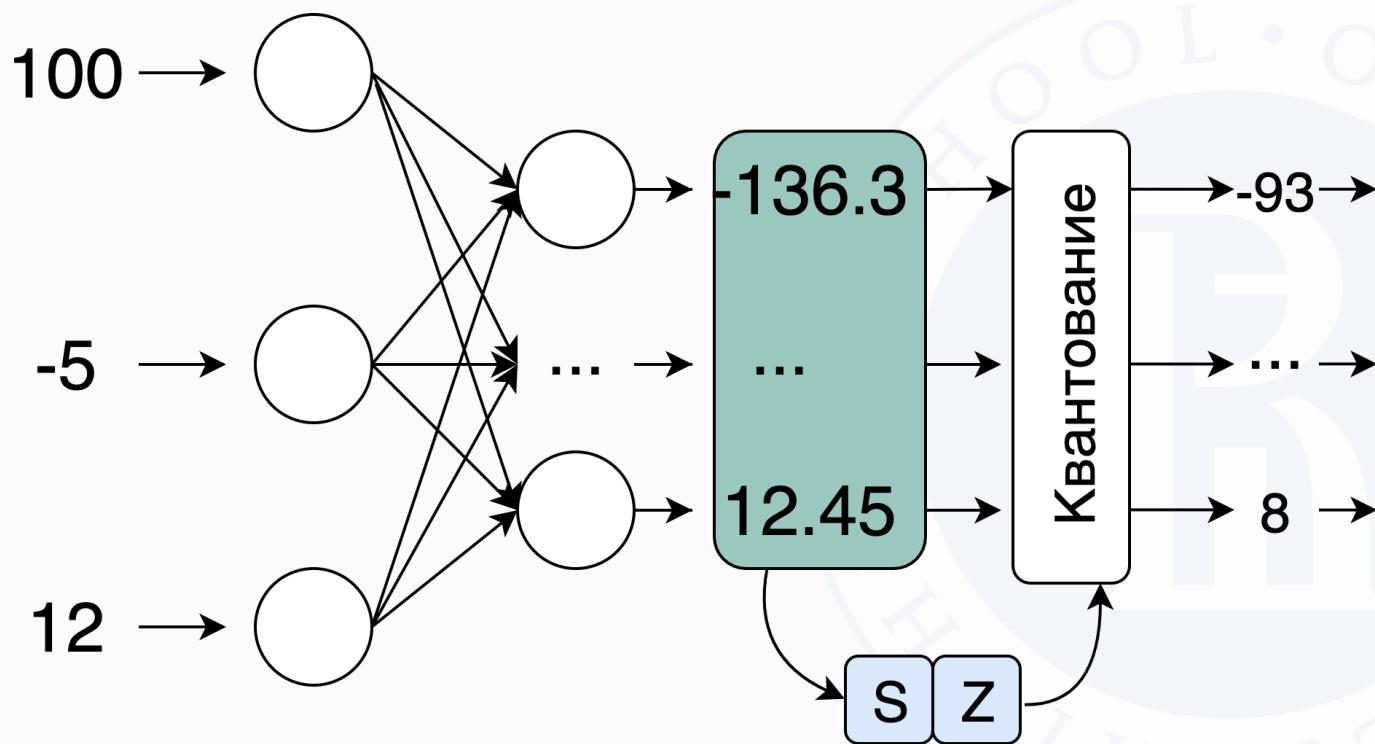
Динамическая квантизация

→ Динамическая квантизация — прием, когда активация квантуется **на лету** для **каждого** входа



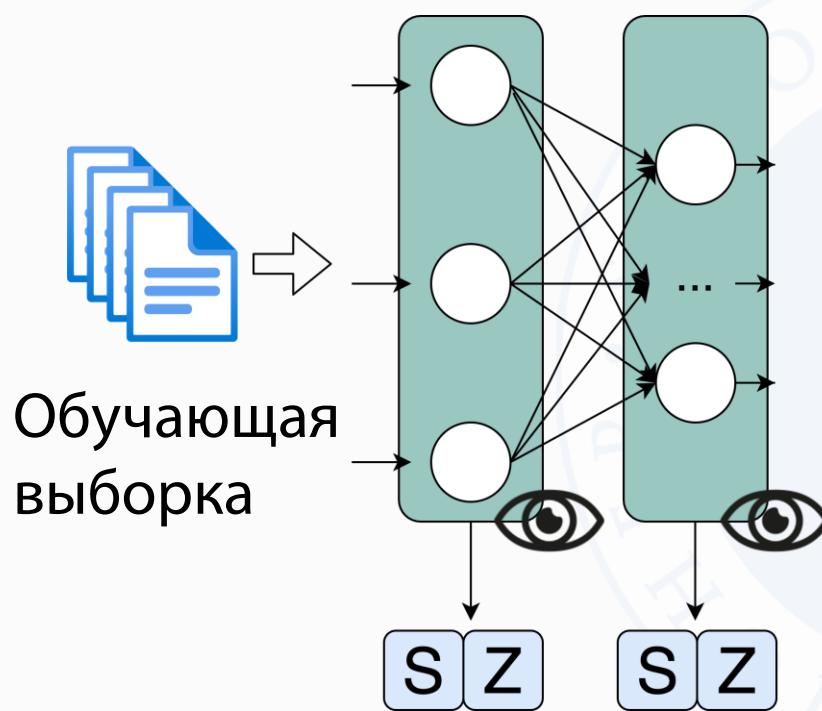
Динамическая квантизация

- Динамическая квантизация — прием, когда активация квантуется **на лету** для **каждого** входа
- Все активации на слое **считываются в FLOAT32**, и значения **обратно квантуются** **наилучшим** для входа образом



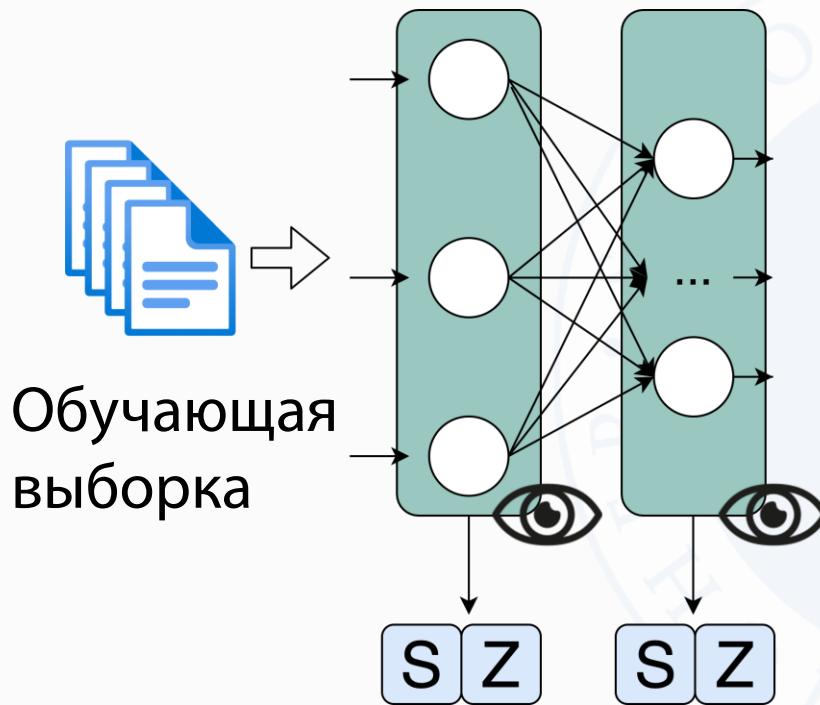
Статическая квантизация

→ Статическая квантизация — прием, когда параметры для активаций рассчитываются один раз по данным из обучающей выборки



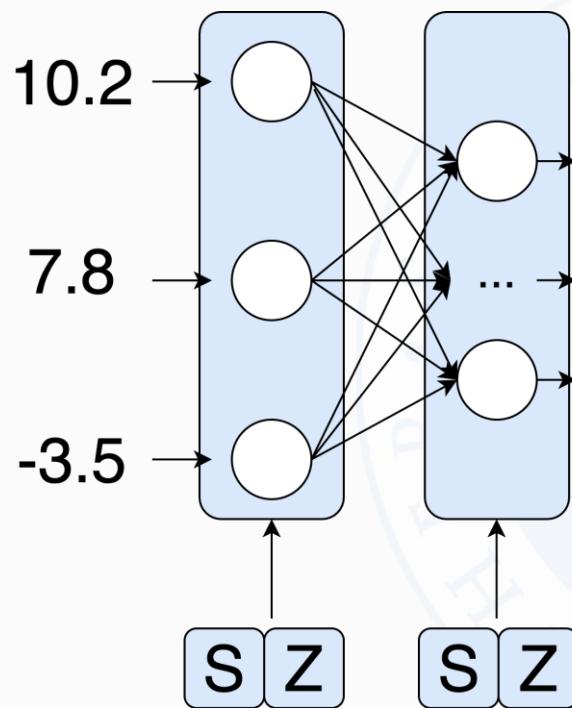
Статическая квантизация

- Статическая квантизация — прием, когда параметры для активаций рассчитываются один раз по данным из обучающей выборки
- На каждом слое мы собираем статистику о том, какие были активации, и из них считаем S и Z



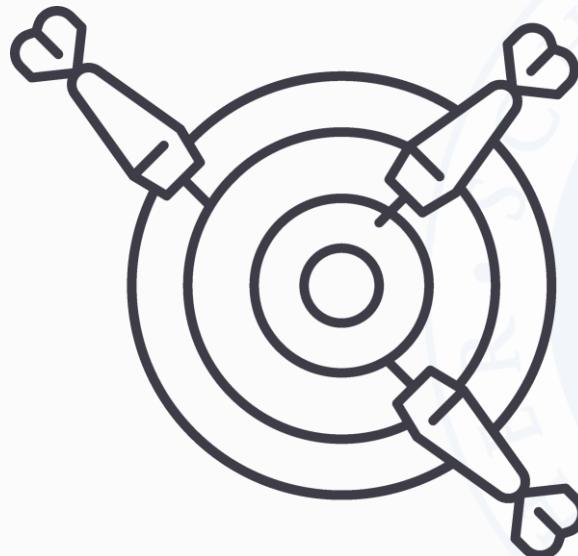
Статическая квантизация

→ После подсчета фиксируем S и Z . Используем их каждый раз для квантования при работе сети. Если новые данные из такого же распределения, что и обучающая выборка, то должно работать хорошо



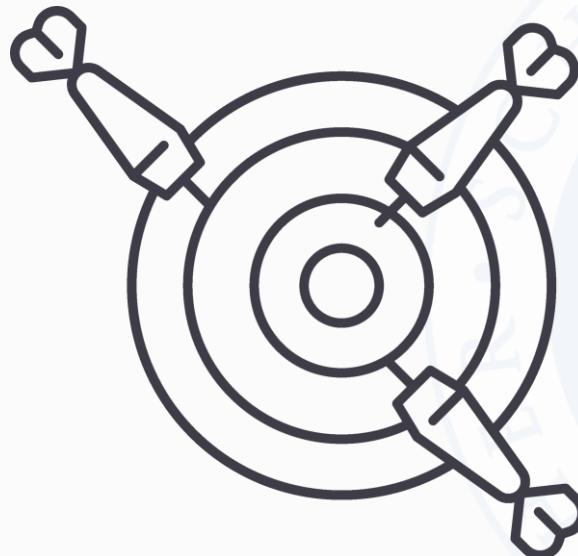
Квантизация в процессе обучения

- Если сети очень глубокие, то использование простых схем квантизации может приводить к накоплению ошибки во время работы сети



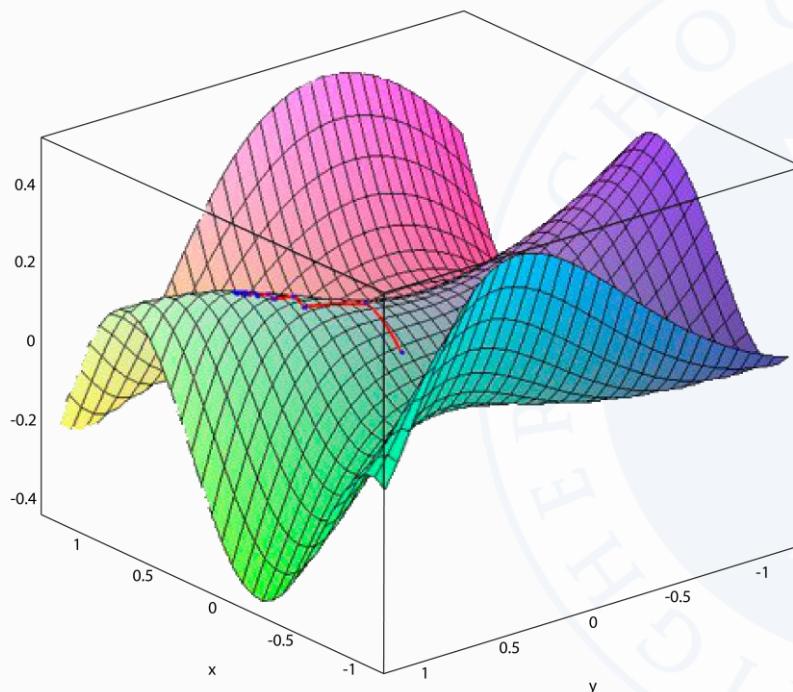
Квантизация в процессе обучения

- Если сети очень глубокие, то использование **простых** схем квантизации может приводить к **накоплению ошибки** во время работы сети
- Чтобы попытаться это **исправить**, применяют прием квантизации в процессе обучения



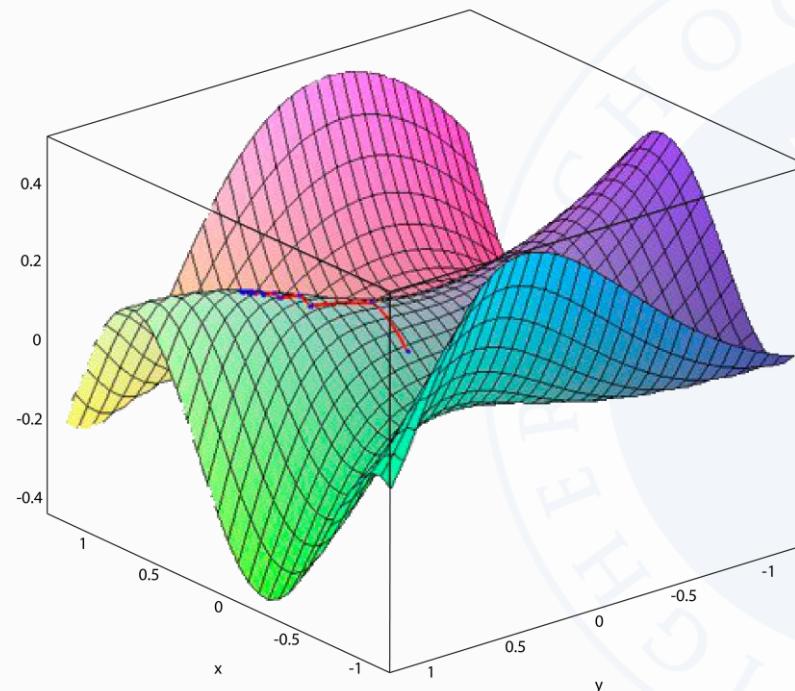
Квантизация в процессе обучения

→ Идея — квантовать веса **после каждого шага градиентного спуска**. Это должно помочь модели сразу **корректировать неточности из-за квантования во время обучения**



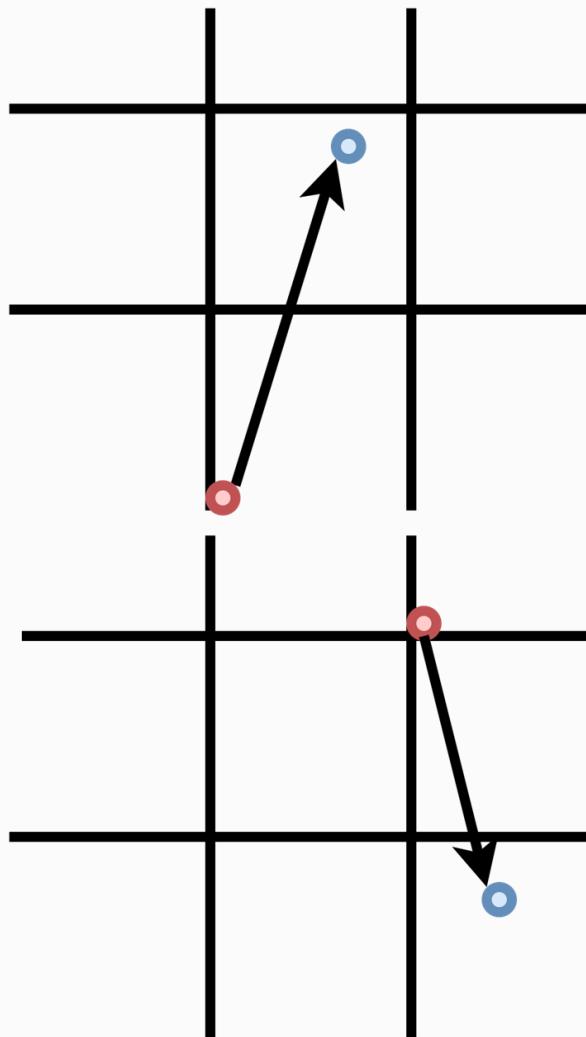
Квантизация в процессе обучения

- Идея — квантовать веса **после каждого шага градиентного спуска**. Это должно помочь модели сразу **корректировать неточности из-за квантования во время обучения**
- Важно отметить, что сам градиент и шаг считается **честно** в **FLOAT32**, чтобы улучшить сходимость

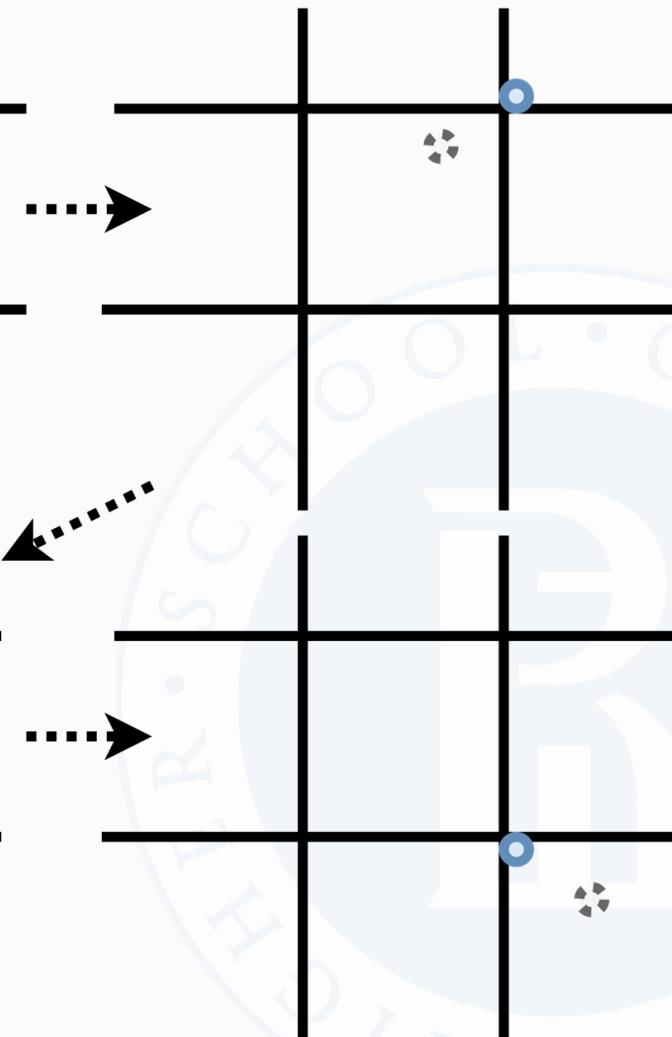


Квантизация в процессе обучения

Шаг спуска



Квантование



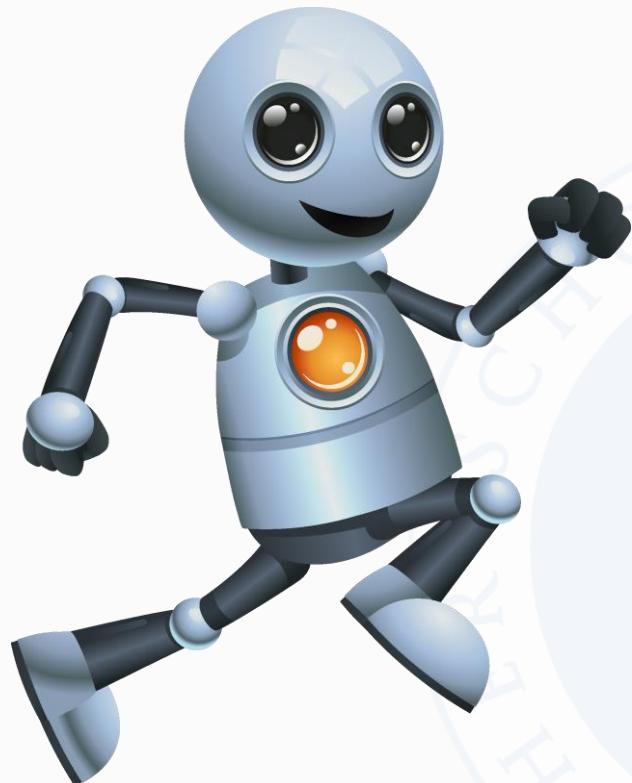
Реальные примеры

→ Ниже приведена таблица с временем и точностью работы некоторых популярных архитектур после квантизации

Модель	Ускорение	Оригинальная точность	Точность после квантизации
googlenet	x2.7	73.45%	73.25%
mobilenet_v2	x2.6	72.40%	73.15%
resnet18	x1.6	70.00%	69.45%
resnet50	x2.7	76.95%	76.60%

Итог

→ Квантизация — класс несложных методов, которые могут значительно уменьшить размер нейронной сети и ускорить ее вычисление



Запуск на мобильных устройствах



Что такое мобильное устройство

→ Самые очевидные представители мобильных устройств — это непосредственно **телефоны и смартфоны**



Что такое мобильное устройство

→ На 2020 год насчитывается более 14 миллиардов телефонов и смартфонов по всему миру



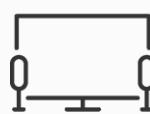
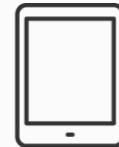
Что такое мобильное устройство



Однако ими список **не ограничивается**



Это и портативные **роботы**, различные **приборы** типа **холодильника** или **плиты**, **аудиосистемы**, **роутеры** и т.д.



Ограничения мобильных устройств

→ Работу всех этих устройств потенциально можно улучшить, добавляя в них алгоритмы искусственного интеллекта



Ограничения мобильных устройств

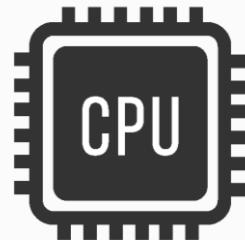
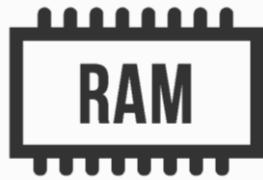
- Работу всех этих устройств потенциально можно улучшить, добавляя в них алгоритмы искусственного интеллекта
- Однако для этого нужно обойти технические ограничения этих устройств



Ограничения мобильных устройств



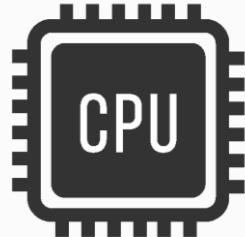
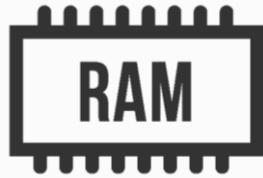
Почти все мобильные устройства обладают следующими ограничениями по сравнению с полноценными серверами:



Ограничения мобильных устройств

→ Почти все мобильные устройства обладают следующими ограничениями по сравнению с полноценными серверами:

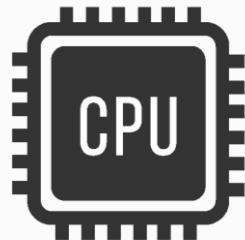
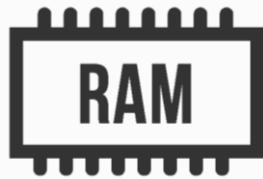
- Малое количество оперативной и долговременной памяти



Ограничения мобильных устройств

→ Почти все мобильные устройства обладают следующими ограничениями по сравнению с полноценными серверами:

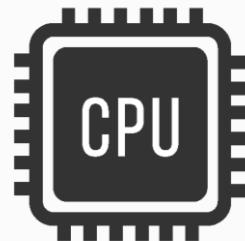
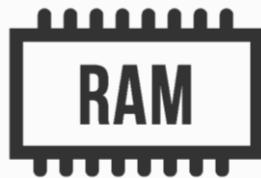
- Малое количество оперативной и долговременной памяти
- Небольшая скорость работы процессора



Ограничения мобильных устройств

→ Почти все мобильные устройства обладают следующими ограничениями по сравнению с полноценными серверами:

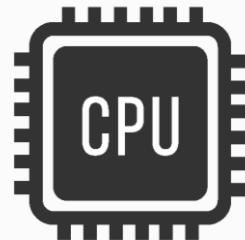
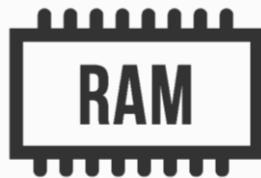
- Малое количество оперативной и долговременной памяти
- Небольшая скорость работы процессора
- Нестабильное или отсутствующее интернет-соединение



Ограничения мобильных устройств

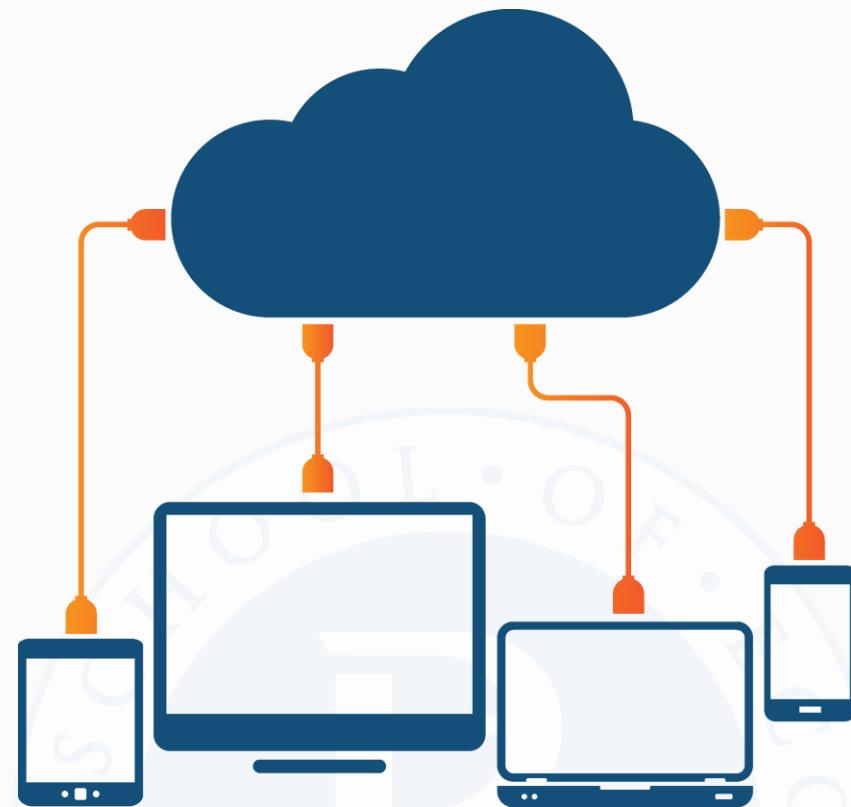
→ Почти все мобильные устройства обладают следующими ограничениями по сравнению с полноценными серверами:

- Малое количество оперативной и долговременной памяти
- Небольшая скорость работы процессора
- Нестабильное или отсутствующее интернет-соединение
- Ограниченнное количество батареи



Ограничения мобильных устройств

→ Нередко эти ограничения можно обойти, запустив вычисления на [серверах](#)



Ограничения мобильных устройств

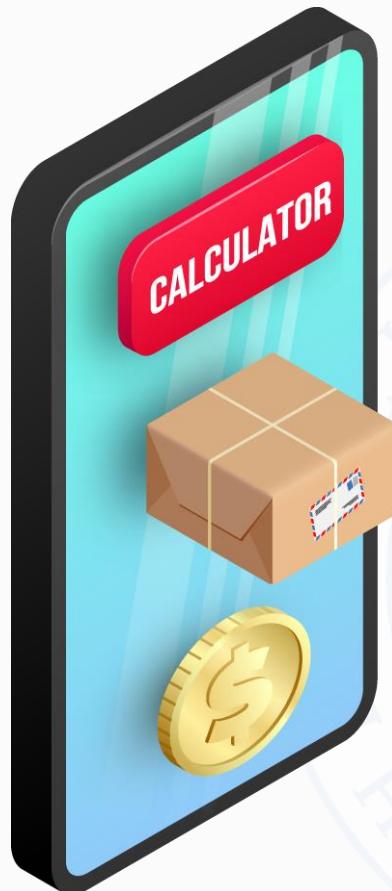
→ Нередко эти ограничения можно обойти, запустив вычисления на [серверах](#)

→ Однако если это слишком затратно или доступ в [интернет ограничен](#), то остается запускать модели прямо на устройстве



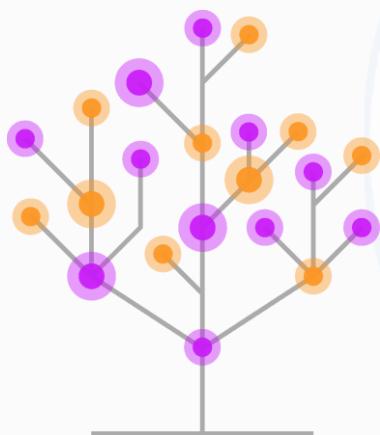
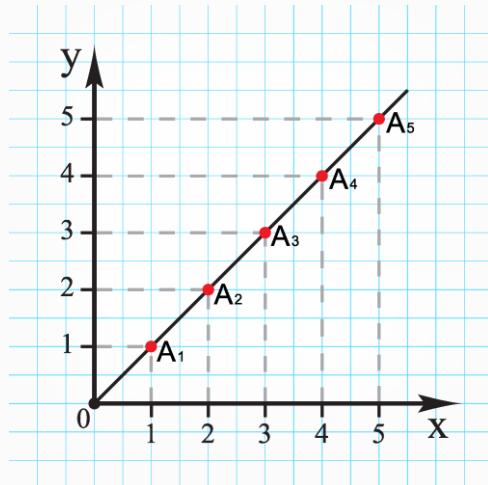
Ограничения мобильных устройств

→ Важно лишь удостовериться, что модели достаточно компактные и используют мало вычислительных операций для работы



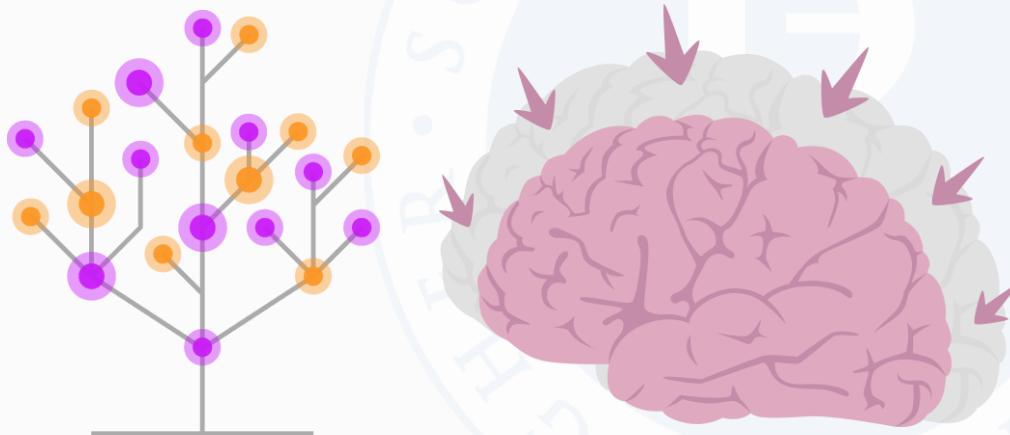
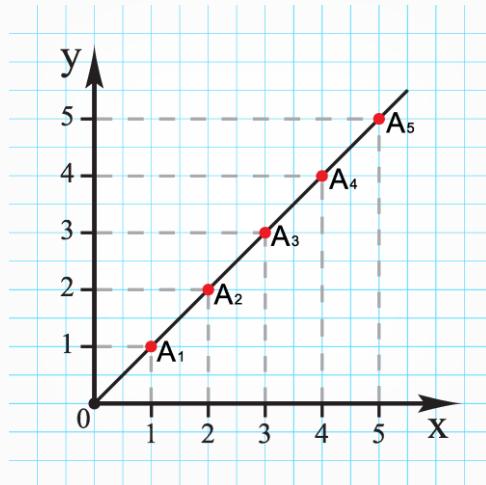
Модели для мобильных устройств

→ Для ряда задач можно использовать **простые** модели, такие как линейные или деревья решений, так как они сами по себе **быстрые** и не требуют много памяти



Модели для мобильных устройств

- Для ряда задач можно использовать **простые** модели, такие как линейные или деревья решений, так как они сами по себе **быстрые** и **не требуют много памяти**
- Если для задачи нужна **нейронная сеть**, то можно использовать **методы оптимизации**, которые мы уже обсудили



Модели для мобильных устройств

→ Однако, помимо оптимизаций существующих алгоритмов, можно использовать специальные архитектуры, спроектированные для работы на мобильных устройствах



MobileNet

→ MobileNet — специальная архитектура сверточных нейронных сетей, разработанная Google

→ Основная идея — использовать специальные «облегченные» сверточные слои



Photo by Juanedo (CC BY 2.0)



Doodle Google by Sarah Harrison

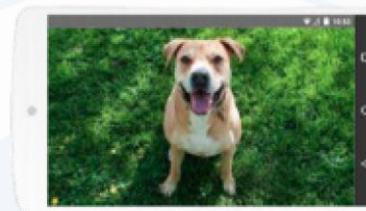


Photo by Harshlight (CC BY 2.0)

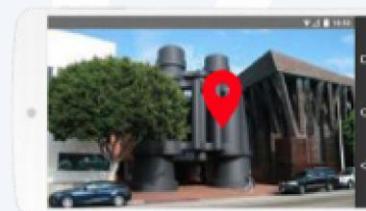
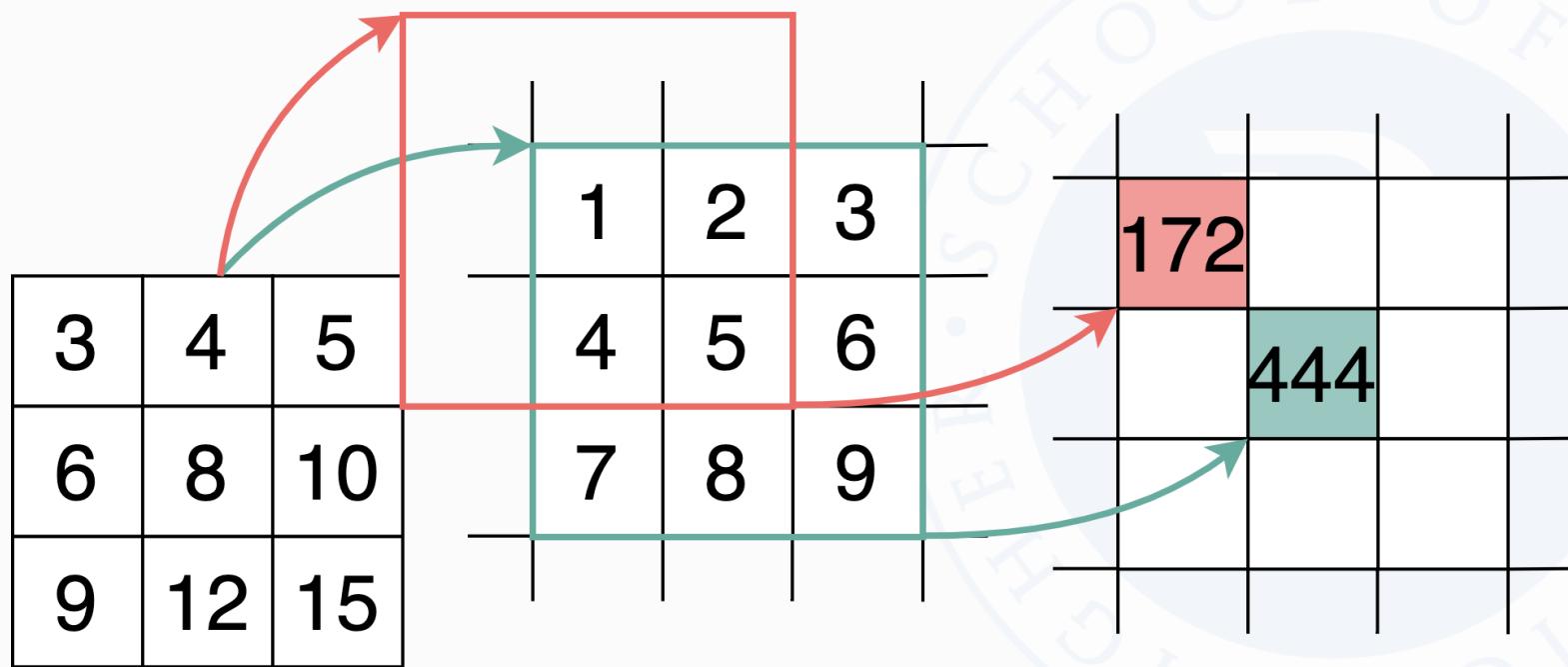


Photo by Sharon Vanderkaay (CC BY 2.0)

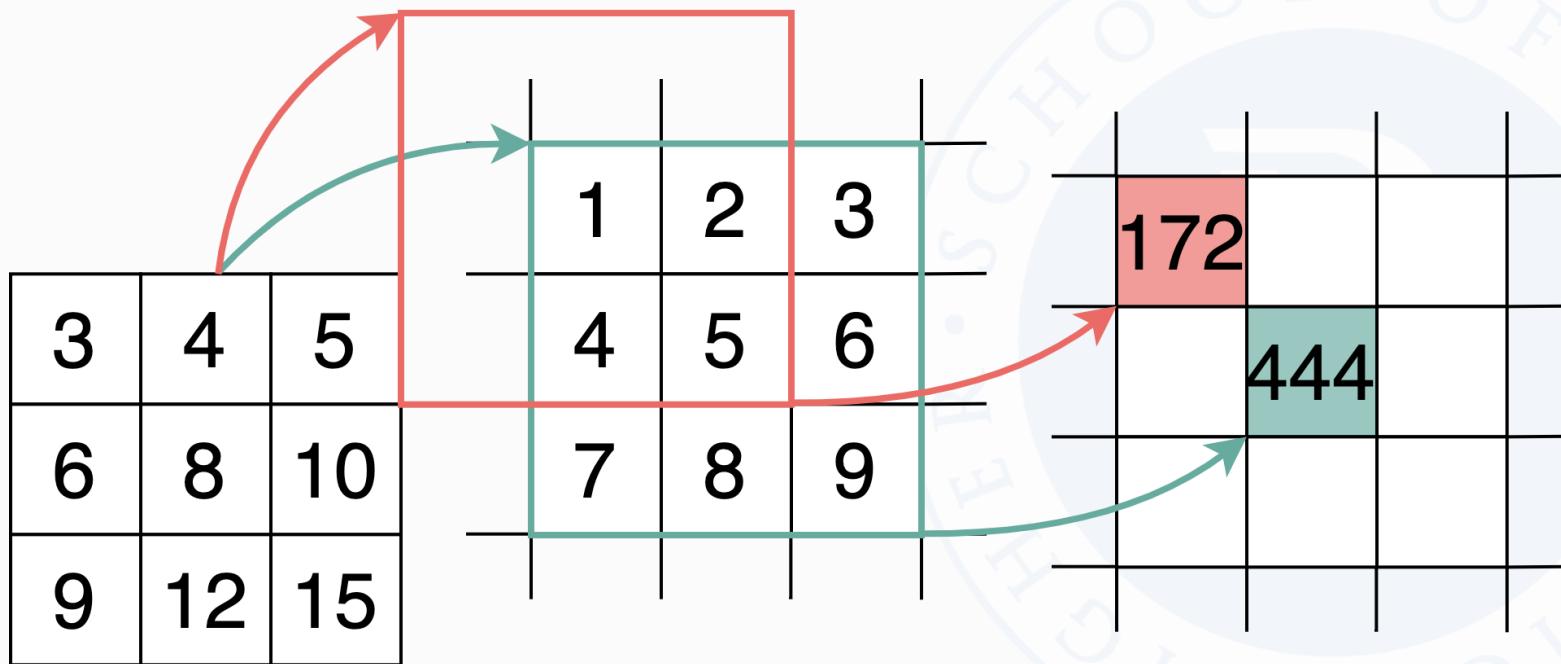
Разделяемые свертки

→ Для наглядности рассмотрим двумерные свертки



Разделяемые свертки

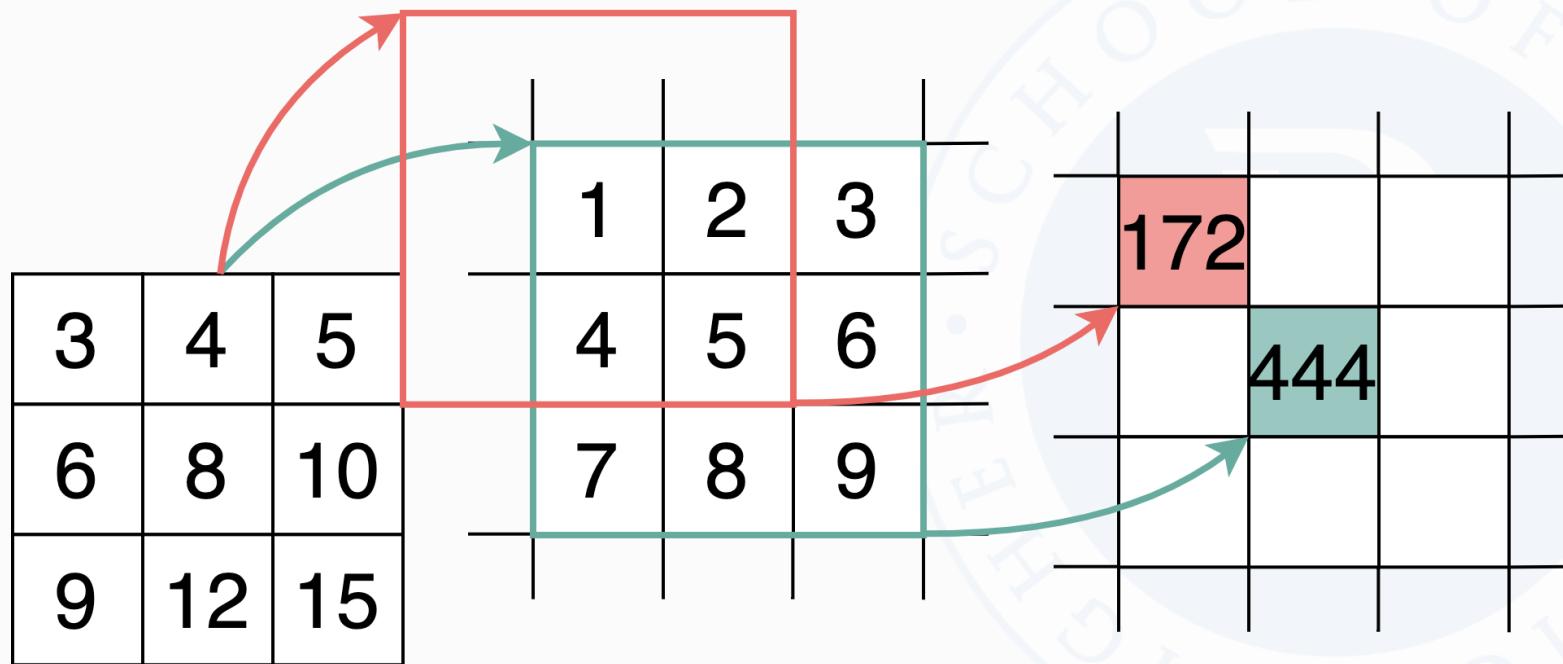
- Для наглядности рассмотрим двумерные свертки
- Чтобы применить свертку $K \cdot K$ к изображению $N \cdot M$, нужно идти скользящим окном $K \cdot K$ по изображению, поэлементно **перемножить** значения в окне со сверткой и **сумму** сложить в **соответствующую ячейку**



Разделяемые свертки

→ Посчитаем количество умножений для расчета.
Для каждой итоговой клетки нужно $K \cdot K$ перемножений.
Всего итоговых клеток $N \cdot M$. Итого:

$$K \cdot K \cdot N \cdot M$$



Разделяемые свертки

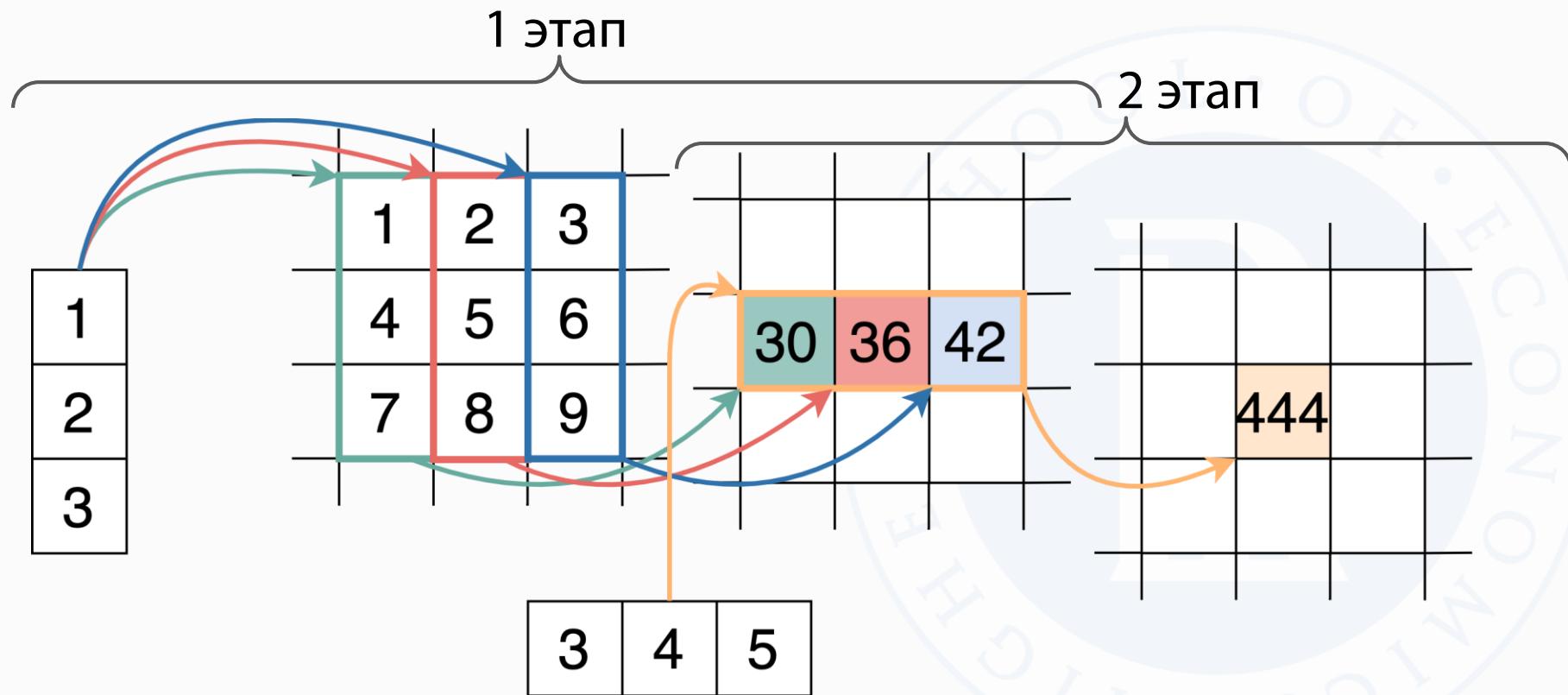
→ Иногда свертку можно разбить на две свертки меньшего размера

→ Например, свертку из примера можно **разбить** на две свертки, каждая размером $K \cdot 1$ и $1 \cdot K$

$$\begin{array}{|c|c|c|} \hline 3 & 4 & 5 \\ \hline 6 & 8 & 10 \\ \hline 9 & 12 & 15 \\ \hline \end{array} = \begin{array}{|c|} \hline 1 \\ \hline 2 \\ \hline 3 \\ \hline \end{array} \times \begin{array}{|c|c|c|} \hline 3 & 4 & 5 \\ \hline \end{array}$$

Разделяемые свертки

→ Тогда сворачивание с изображением можно провести в 2 этапа: вначале свернуть с первой сверткой $K \cdot 1$, и потом результат — со второй $1 \cdot K$



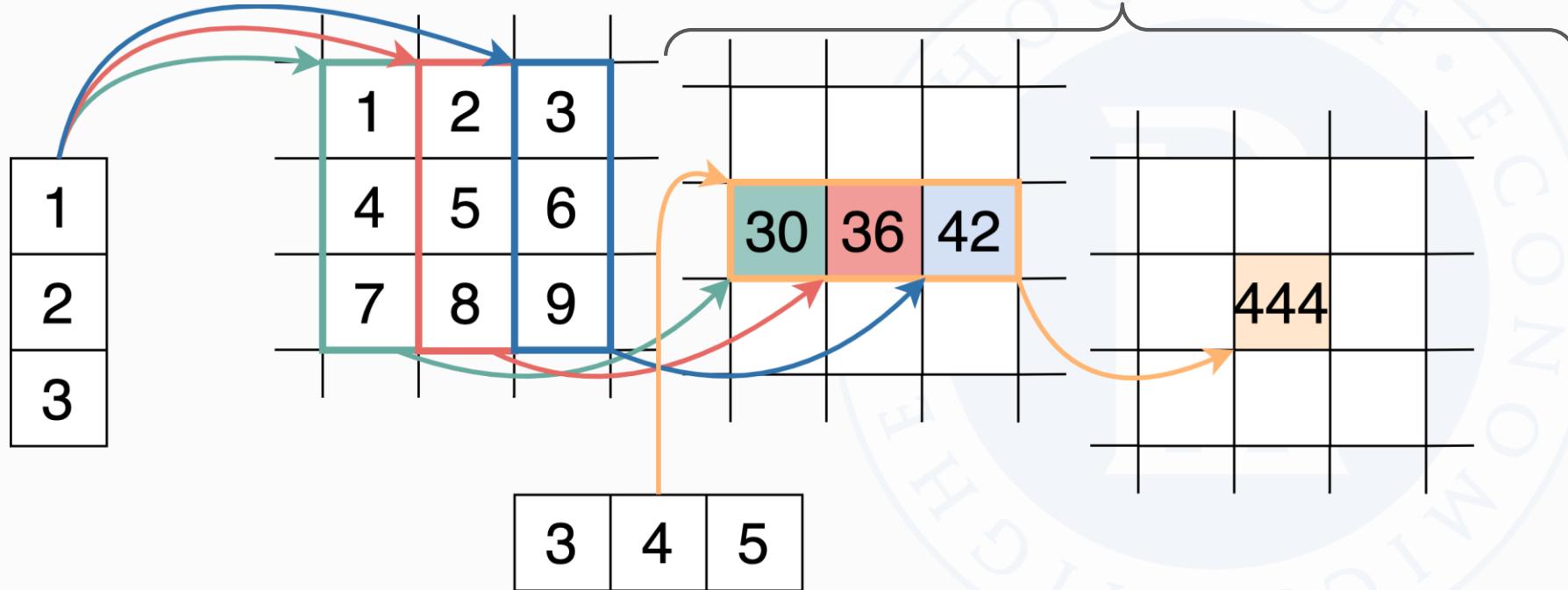
Разделяемые свертки

→ Подсчитаем количество умножений. На 1 этапе мы сделали $1 \cdot K \cdot N \cdot M$, на втором $K \cdot 1 \cdot N \cdot M$. Получается:

$$2 \cdot K \cdot N \cdot M$$

1 этап

2 этап



Разделяемые свертки

→ Подсчитаем количество умножений. На 1 этапе мы сделали $1 \cdot K \cdot N \cdot M$, на втором $K \cdot 1 \cdot N \cdot M$. Получается:

$$2 \cdot K \cdot N \cdot M$$

→ Если $K > 2$, то получается, что мы **сильно уменьшаем** количество требуемых перемножений!

Разделяемые свертки

→ Подсчитаем количество умножений. На 1 этапе мы сделали $1 \cdot K \cdot N \cdot M$, на втором $K \cdot 1 \cdot N \cdot M$. Получается:

$$2 \cdot K \cdot N \cdot M$$

- Если $K > 2$, то получается, что мы **сильно уменьшаем** количество требуемых перемножений!
- В нашем примере $K = 3$, а значит вместо $9 \cdot N \cdot M$ мы сделали всего $6 \cdot N \cdot M$

Разделяемые свертки

→ Подсчитаем количество умножений. На 1 этапе мы сделали $1 \cdot K \cdot N \cdot M$, на втором $K \cdot 1 \cdot N \cdot M$. Получается:

$$2 \cdot K \cdot N \cdot M$$

→ Если $K > 2$, то получается, что мы сильно уменьшаем количество требуемых перемножений!

→ В нашем примере $K = 3$, а значит вместо $9 \cdot N \cdot M$ мы сделали всего $6 \cdot N \cdot M$

→ Такие свертки получили название Separable Convolutions

Разделяемые свертки

→ Таким образом, разделение сверток на более маленькие вполне **оправдано** — результат может быть **не хуже**, а **скорость** — **выше**

$$\begin{array}{|c|c|c|} \hline 3 & 4 & 5 \\ \hline 6 & 8 & 10 \\ \hline 9 & 12 & 15 \\ \hline \end{array} = \begin{array}{|c|} \hline 1 \\ \hline 2 \\ \hline 3 \\ \hline \end{array} \times \begin{array}{|c|c|c|} \hline 3 & 4 & 5 \\ \hline \end{array}$$

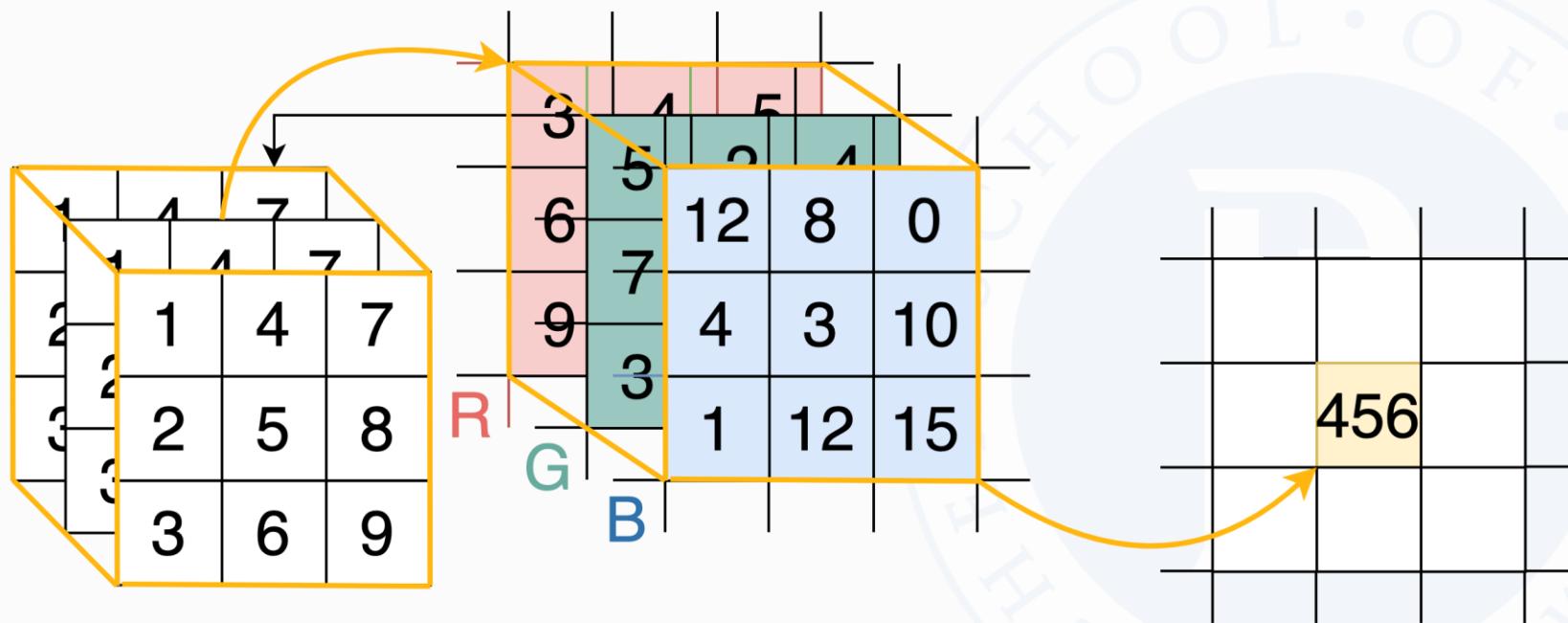
Разделяемые свертки

- Аналогичная идея используется для трехмерных сверток, которые являются основой современных сверточных сетей
- Изображение — трехмерная матрица, где третья размерность — это каналы изображения (например, RGB)

	3	4	5	
	6	5	2	4
	7		12	8
	9		0	
R	3			
G		4	3	10
B	3	1	12	15

Разделяемые свертки

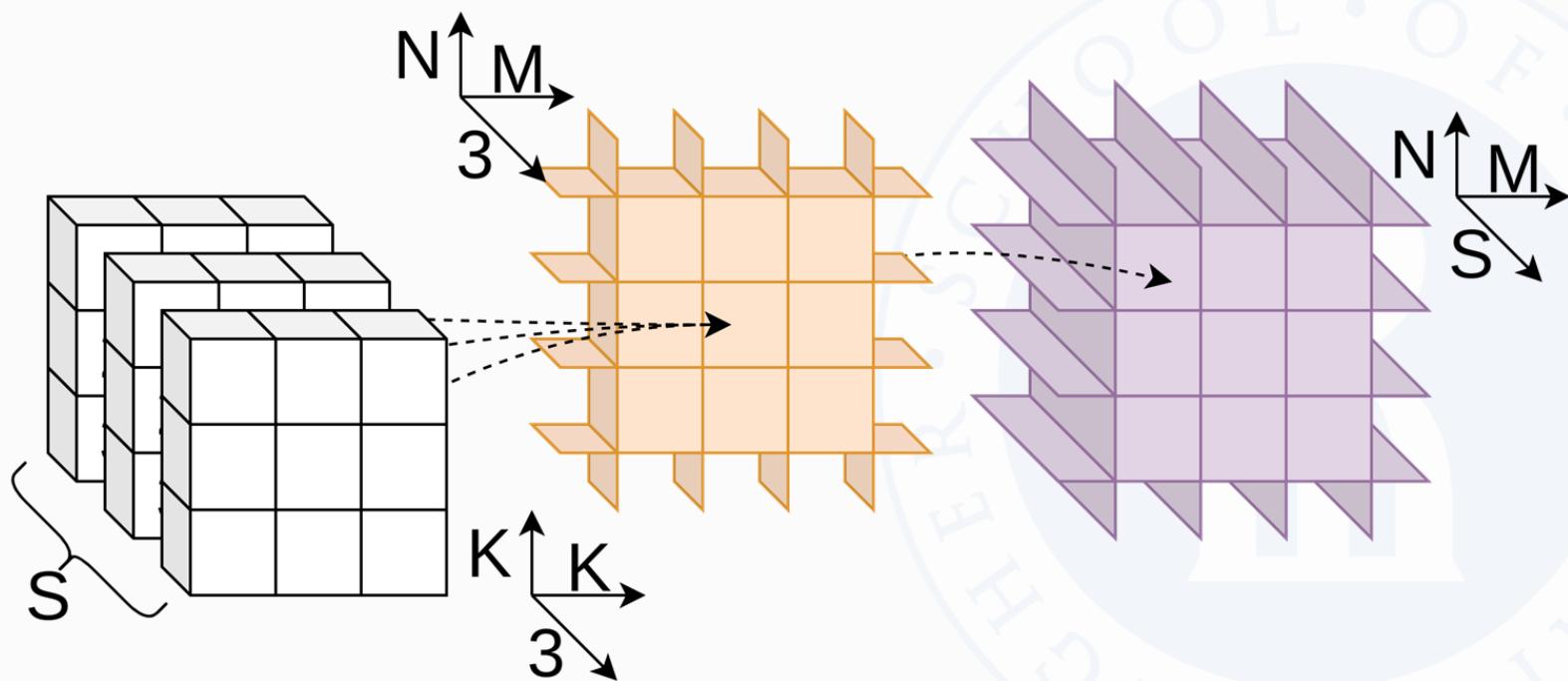
→ У изображения D каналов, значит, количество элементов $D \cdot N \cdot M$. Соответственно, свертка — это тоже трехмерная матрица размера $D \cdot K \cdot K$



Разделяемые свертки

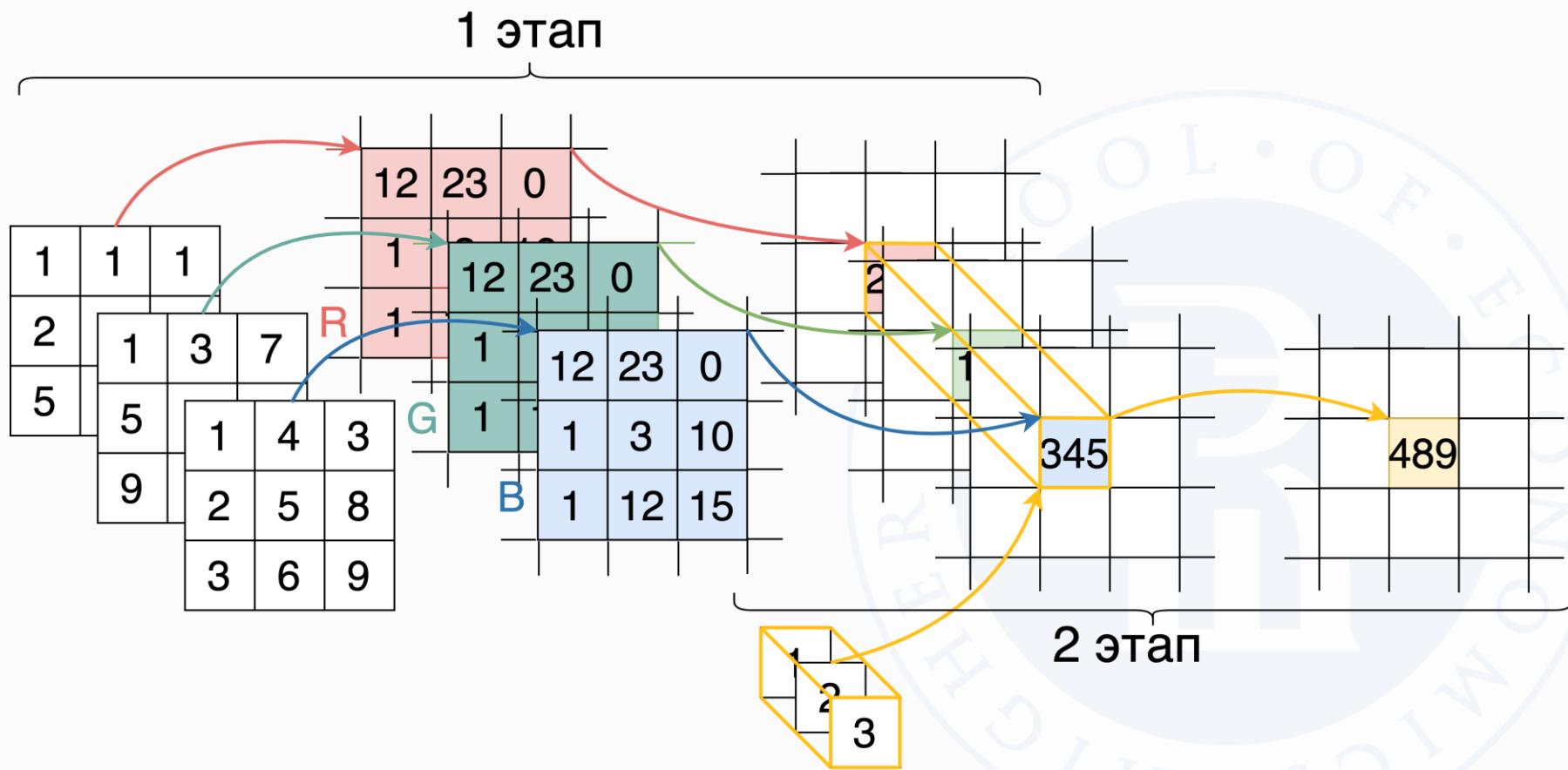
→ Если мы хотим получить S каналов в результате, то нужно использовать S таких сверток. Итоговое количество умножений:

$$D \cdot K \cdot K \cdot M \cdot N \cdot S$$



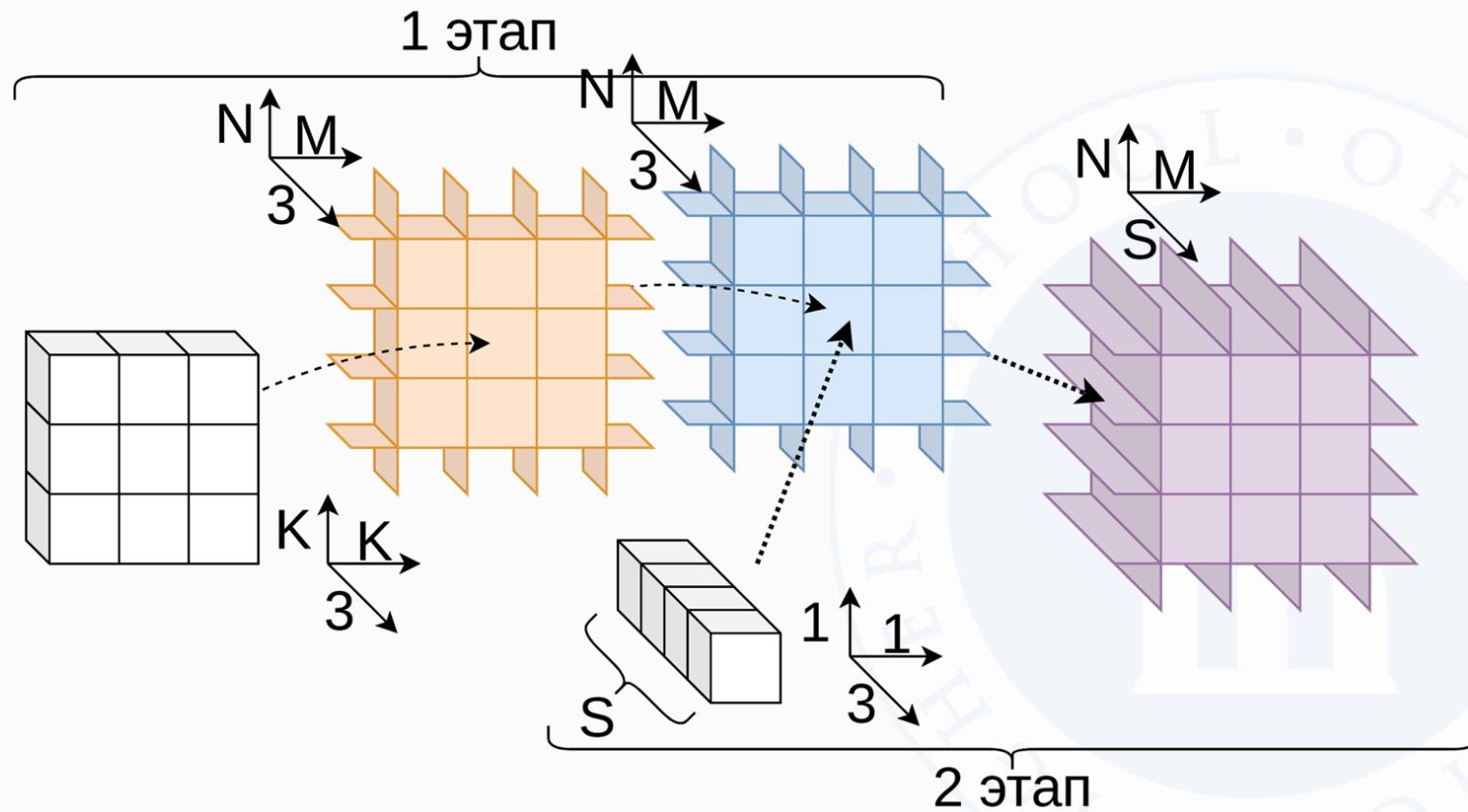
Разделяемые свертки

→ «Разделим» теперь свертки по каналам: на первом этапе посчитаем соответствующую свертку $1 \cdot K \cdot K$ для каждого канала, на втором — посчитаем свертку $D \cdot 1 \cdot 1$



Разделяемые свертки

→ Чтобы получить S каналов на выходе, применим S сверток размера $D \cdot 1 \cdot 1$. На первом этапе — $K \cdot K \cdot N \cdot M \cdot D$ умножений, на втором — $D \cdot N \cdot M \cdot S$

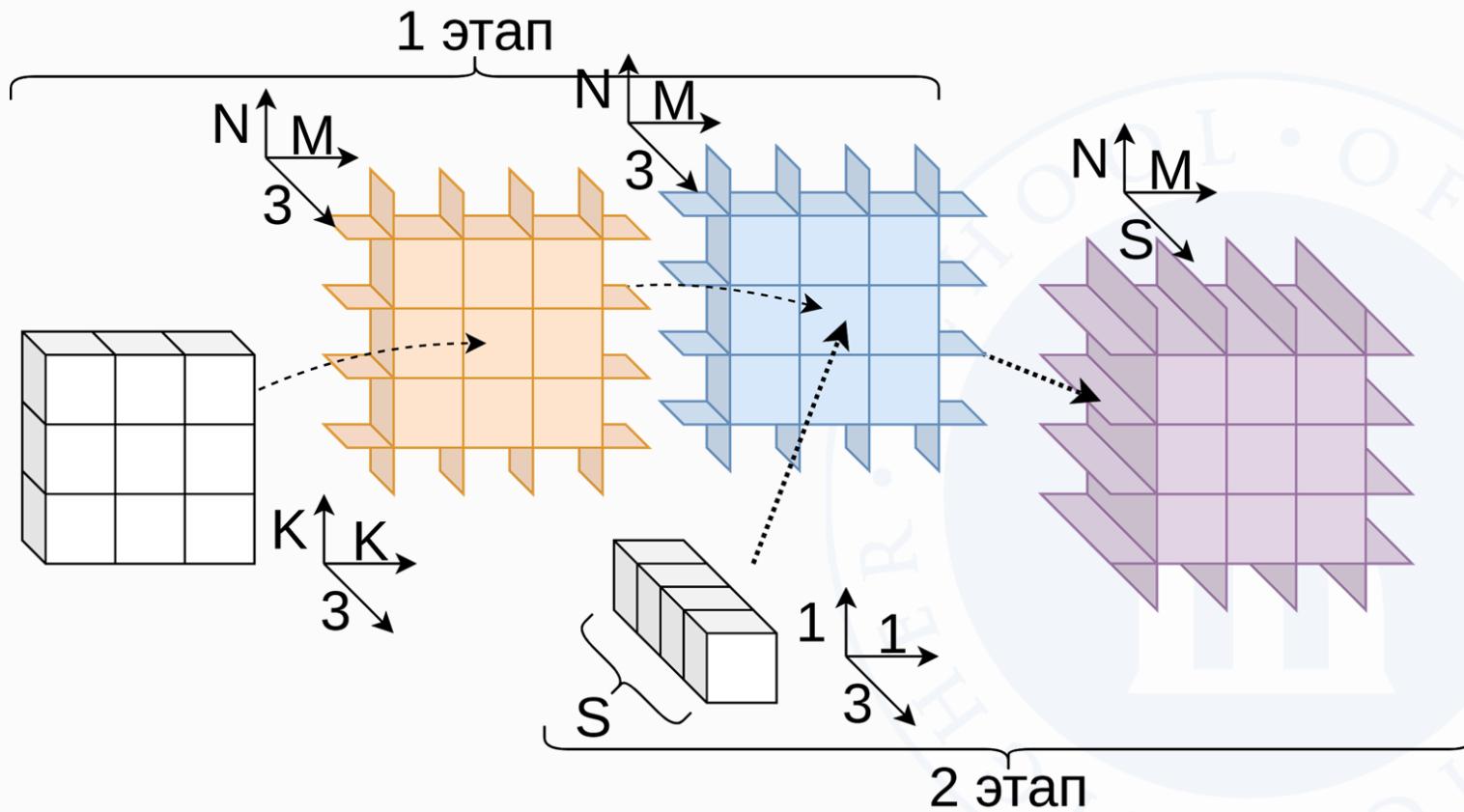


Разделяемые свертки



Итого: $K \cdot K \cdot N \cdot M \cdot D + D \cdot N \cdot M \cdot S =$

$$N \cdot M \cdot D \cdot (K \cdot K + S)$$



Разделяемые свертки

→ Итого: $K \cdot K \cdot N \cdot M \cdot D + D \cdot N \cdot M \cdot S =$

$$N \cdot M \cdot D \cdot (K \cdot K + S)$$

→ Для того чтобы ощутить, насколько быстрее получилось, возьмем конкретные цифры

$$N = 256, M = 256, S = 64, K = 3, D = 10$$

Разделяемые свертки

→ Итого: $K \cdot K \cdot N \cdot M \cdot D + D \cdot N \cdot M \cdot S =$

$$N \cdot M \cdot D \cdot (K \cdot K + S)$$

→ Для того чтобы ощутить, насколько быстрее получилось, возьмем конкретные цифры

$$N = 256, M = 256, S = 64, K = 3, D = 10$$

→ Обычные свертки: $103 \cdot 3 \cdot 256 \cdot 256 \cdot 64 = 377$ миллионов

→ Разделимые: $256 \cdot 256 \cdot 10 \cdot (3 \cdot 3 + 64) = 47$ миллионов

→ Почти в 8 раз меньше!

Разделяемые свертки

- Количество параметров также уменьшилось
- Для классических сверток необходимо хранить $K \cdot K \cdot D \cdot S$ параметров, а для разделяемых — всего $K \cdot K \cdot D + D \cdot S$

Разделяемые свертки

- Количество параметров также уменьшилось
- Для классических сверток необходимо хранить $K \cdot K \cdot D \cdot S$ параметров, а для разделяемых — всего $K \cdot K \cdot D + D \cdot S$
- Для нашего примера

$$N = 256, M = 256, S = 64, K = 3, D = 10$$

Разделяемые свертки

- Количество параметров также уменьшилось
- Для классических сверток необходимо хранить $K \cdot K \cdot D \cdot S$ параметров, а для разделяемых — всего $K \cdot K \cdot D + D \cdot S$

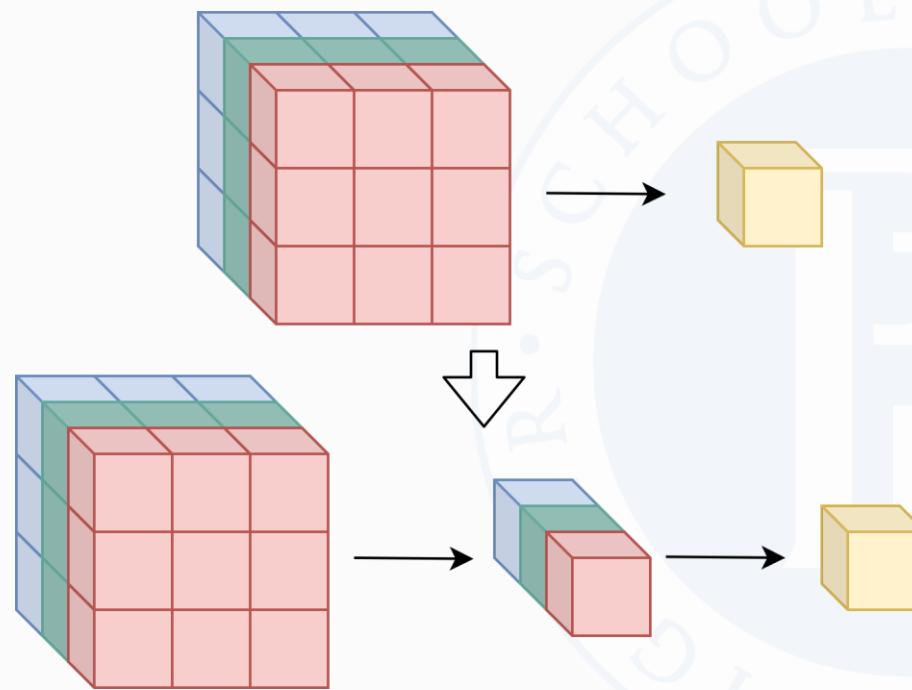
- Для нашего примера

$$N = 256, M = 256, S = 64, K = 3, D = 10$$

- Обычные свертки: $3 \cdot 3 \cdot 10 \cdot 64 = 5760$ параметров
- Разделимые: $3 \cdot 3 \cdot 10 + 10 \cdot 64 = 730$ параметров
- Также почти в 8 раз меньше!

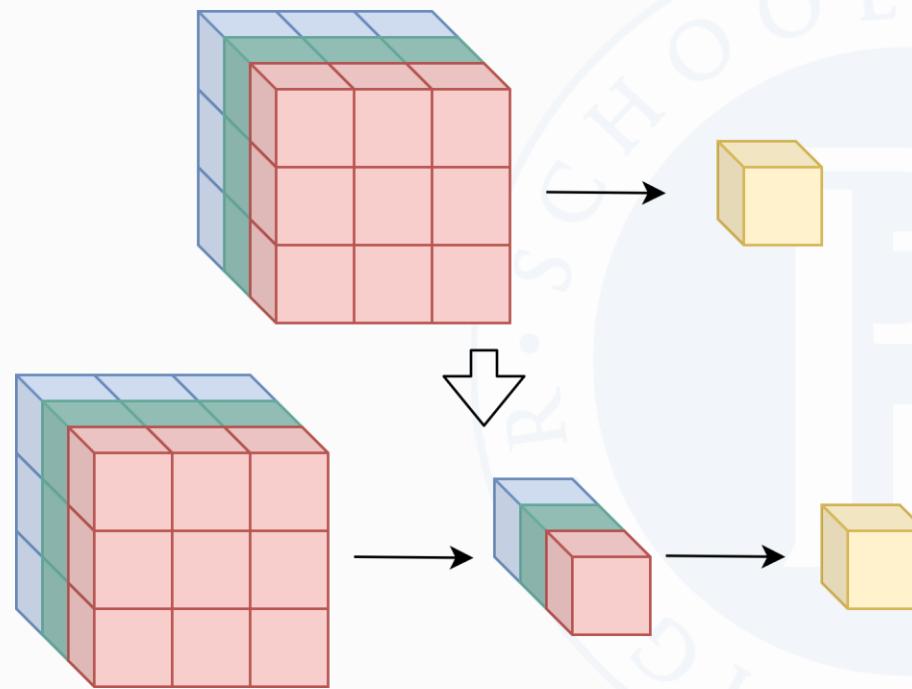
Разделяемые свертки

→ Такие свертки получили название Depthwise Separable Convolutions



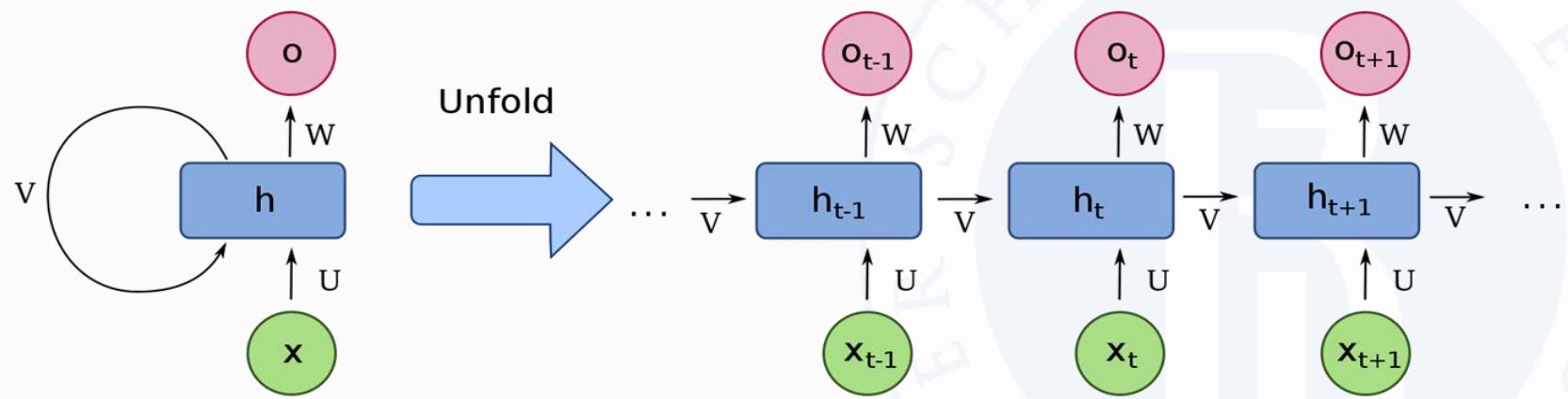
Разделяемые свертки

- Такие свертки получили название Depthwise Separable Convolutions
- Используя специальный тип сверточек, MobileNet успешно может решать сложные задачи прямо на мобильном устройстве



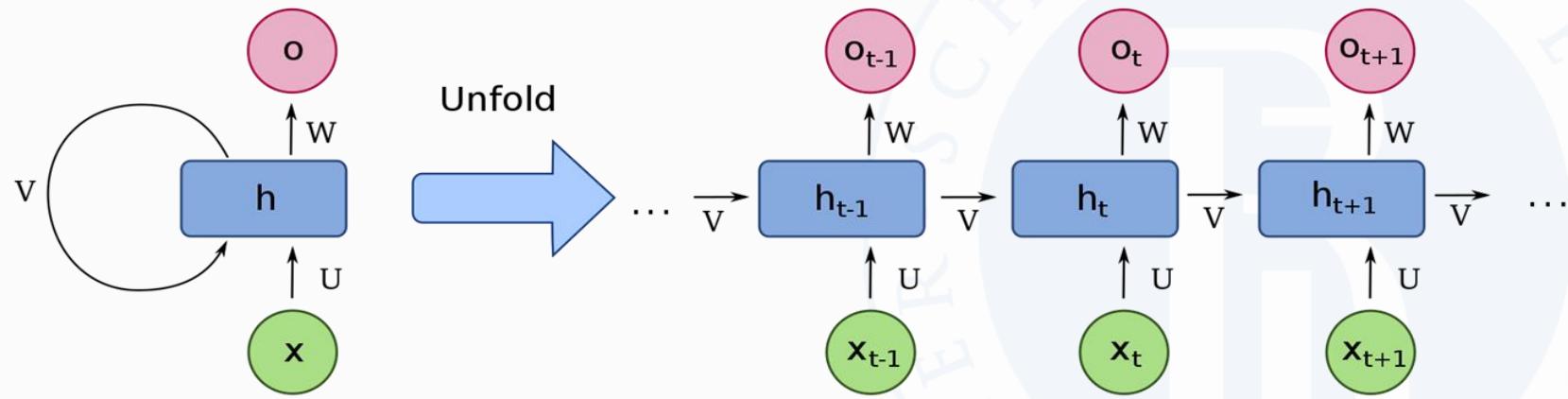
LightRNN

→ Другой пример адаптированной архитектуры — LightRNN, которая позволяет строить компактные **рекуррентные** сети



LightRNN

- Другой пример адаптированной архитектуры — LightRNN, которая позволяет строить компактные **рекуррентные** сети
- Основная идея — использовать **специальное кодирование** слов для сети, использующее **общие веса**



LightRNN

→ У любого типа RNN есть проблема — для **каждого** возможного слова необходимо **хранить** вектор представления

Слово	Вектор
Понедельник	(1.2, ..., 7.1)
Вторник	(3.2, ..., 2.1)
...	...
Погода	(1.7, ..., 4.3)

LightRNN

- У любого типа RNN есть проблема — для каждого возможного слова необходимо хранить вектор представления
- Это вызывает проблемы с размером модели, а также со скоростью ее обучения

Слово	Вектор
Понедельник	(1.2, ..., 7.1)
Вторник	(3.2, ..., 2.1)
...	...
Погода	(1.7, ..., 4.3)

LightRNN

→ Предположим, что размер словаря — 10 миллионов слов, и мы используем 1024 числа для кодирования слова

Слово	Вектор
Понедельник	(1.2, ..., 7.1)
Вторник	(3.2, ..., 2.1)
...	...
Погода	(1.7, ..., 4.3)

LightRNN

→ Предположим, что размер словаря — 10 миллионов слов, и мы используем 1024 числа для кодирования слова

→ Для хранения обученной модели нам потребуется около 40 Гб

Слово	Вектор
Понедельник	(1.2, ..., 7.1)
Вторник	(3.2, ..., 2.1)
...	...
Погода	(1.7, ..., 4.3)

LightRNN

- Предположим, что размер словаря — **10 миллионов** слов, и мы используем **1024 числа** для кодирования слова
- Для хранения обученной модели нам потребуется около **40 Гб**
- На **телефоне** такое точно не запустить!

Слово	Вектор
Понедельник	(1.2, ..., 7.1)
Вторник	(3.2, ..., 2.1)
...	...
Погода	(1.7, ..., 4.3)

→ Основной прием, позволяющий с этим справиться, называется **2С-представление** — 2-х компонентное объединенное представление векторов. Каждому слову будут соответствовать **два компонента** — x_r и x_c

Слово	Компоненты
Понедельник	$x_r=(1.2, \dots, 3.4)$, $x_c=(5.6, \dots, 2.3)$
Вторник	$x_1=(2.2, \dots, 3.6)$, $x_2=(3.6, \dots, 0.0)$
...	...
Погода	$x_1=(14.0, \dots, 6.4)$, $x_2=(5.4, \dots, 2.1)$

LightRNN

→ Чтобы назначить словам их компоненты, мы составляем квадратную таблицу. Каждой строке соответствует какой-то первый компонент, каждой колонке — второй компонент

	x_{c1}	x_{c2}	...	x_{cN}
x_{r1}				
x_{r2}				
...				
x_{rN}				

LightRNN

→ Внутри таблицы расположим **случайным образом** все слова.
Положение слова в таблице **определяет его первый и второй компоненты** для представления

	x_{c1}	x_{c2}	...	x_{cN}
x_{r1}	Понедельник	Вторник
x_{r2}	Месяц	Собака
...
x_{rN}	Погода

LightRNN

→ Например, в нашей таблице слово «Месяц» имеет представление (x_{r2} , x_{c1})

	x_{c1}	x_{c2}	...	x_{cN}
x_{r1}	Понедельник	Вторник
x_{r2}	Месяц	Собака
...
x_{rN}	Погода

LightRNN

→ Важно заметить, что если сторона таблицы N, то каждый компонент является общим для N слов

	x_{c1}	x_{c2}	...	x_{cN}
x_{r1}	Понедельник	Вторник
x_{r2}	Месяц	Собака
...
x_{rN}	Погода

LightRNN

- Важно заметить, что если сторона таблицы N , то каждый компонент является общим для N слов
- Однако комбинация первого и второго компонентов уникальна для каждого слова!

	x_{c1}	x_{c2}	...	x_{cN}
x_{r1}	Понедельник	Вторник
x_{r2}	Месяц	Собака
...
x_{rN}	Погода

LightRNN

→ Сколько же теперь нам требуется векторов для кодирования **всех слов** в словаре?

LightRNN

- Сколько же теперь нам требуется векторов для кодирования **всех слов** в словаре?
- Если всего слов в словаре **M**, и мы разместили их в квадратной матрице со стороной **N**, то значит

$$N = \sqrt{M}$$

LightRNN

- Сколько же теперь нам требуется векторов для кодирования **всех слов** в словаре?
- Если всего слов в словаре **M**, и мы разместили их в квадратной матрице со стороной **N**, то значит
$$N = \sqrt{M}$$
- Получается, что нам требуется всего $2\sqrt{M}$ векторов для кодирования вместо изначальных **M**

LightRNN

→ Для нашего примера с **10 миллионами** слов в словаре и для размера вектора в **1024 элемента** нам потребуется всего **24 Мб** вместо **40 Гб**!



LightRNN

- Для нашего примера с **10 миллионами** слов в словаре и для размера вектора в **1024 элемента** нам потребуется всего **24 Мб** вместо **40 Гб**!
- Такую модель теперь вполне можно запускать на небольших устройствах



Особенности кодирования

→ Может возникнуть правомерный вопрос — а не **упадет ли качество из-за такого кодирования?**



Особенности кодирования

- Может возникнуть правомерный вопрос — а не **упадет ли качество из-за такого кодирования?**
- Если данных для обучения **очень много**, то даже такой способ кодирования **способен обеспечить хорошее качество**



Особенности кодирования

- Однако авторы также предлагают **дополнительный трюк**, чтобы улучшить модель
- После раунда обучения можно **пересортировать слова** в таблице, чтобы **качество кодирования увеличилось**

	x_{c1}	x_{c2}	...	x_{cN}
x_{r1}	Понедельник	Вторник
x_{r2}	Месяц	Собака
...
x_{rN}	Погода

The diagram illustrates a circular permutation of words in the second and third columns of the table. A large circle is drawn around the second and third columns. An arrow points from the word 'Вторник' (Second) in the third column back to the second column, and another arrow points from the word 'Собака' (Dog) in the second column back to the third column, forming a closed loop.

Особенности кодирования

→ Результаты показывают, что после такой пересортировки качество может стать даже лучше, так как близкие по смыслу слова теперь сгруппируются по колонкам и столбцам

	x_{c1}	x_{c2}	...	x_{cN}
x_{r1}	Зеленый	Синий
x_{r2}	Трава	Река
...
x_{rN}	Метр

Инструменты для запуска

- В индустрии уже существует большое количество инструментов для запуска моделей машинного обучения на мобильных устройствах. Например:
- CoreML — для запуска на iOS
 - ML Kit — для запуска на Android
 - Tensorflow JS — для запуска в браузере



Итоги

→ Таким образом, искусственный интеллект на портативных девайсах уже **не является чем-то невероятным** и этот подход можно **активно использовать** для разработки приложений на базе машинного обучения

