

Лекция 1

Обучение с учителем

Разработка нейросетевых систем

Материалы лекции



This NVIDIA DLI Certificate has been awarded to

Anton Kanev

for the successful completion of
Fundamentals of Deep Learning

Will Ramey



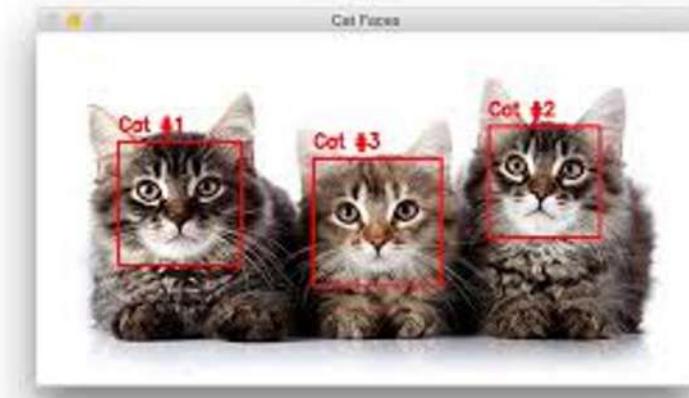
В лекции использованы материалы DLI Teaching Kit Deep Learning лицензированные компанией NVIDIA и New York University под лицензией [Creative Commons Attribution-NonCommercial 4.0 International License](#).

Машинное обучение

- Машинное обучение – это возможность обучать компьютер без детального программирования
- Для обучения компьютера используются наборы примеров для решения задач, которые сложно представить в программном коде

First Name

Last Name



Распознавание лиц/угроз на изображении

- Различные методы распознавания лиц

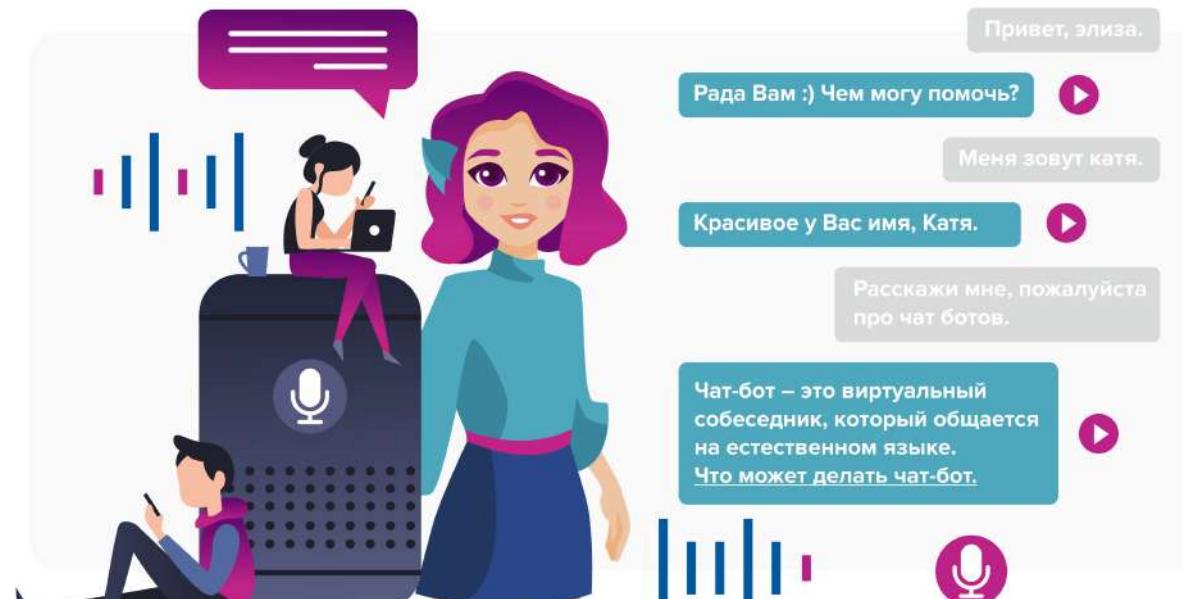
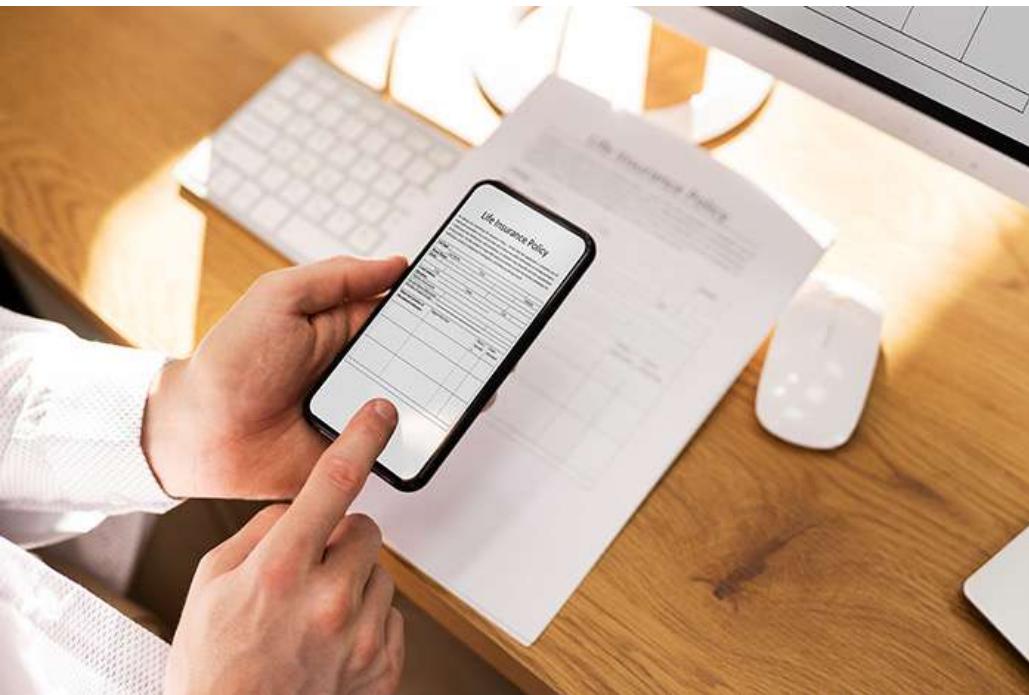


Распознавание и локализация yolo:

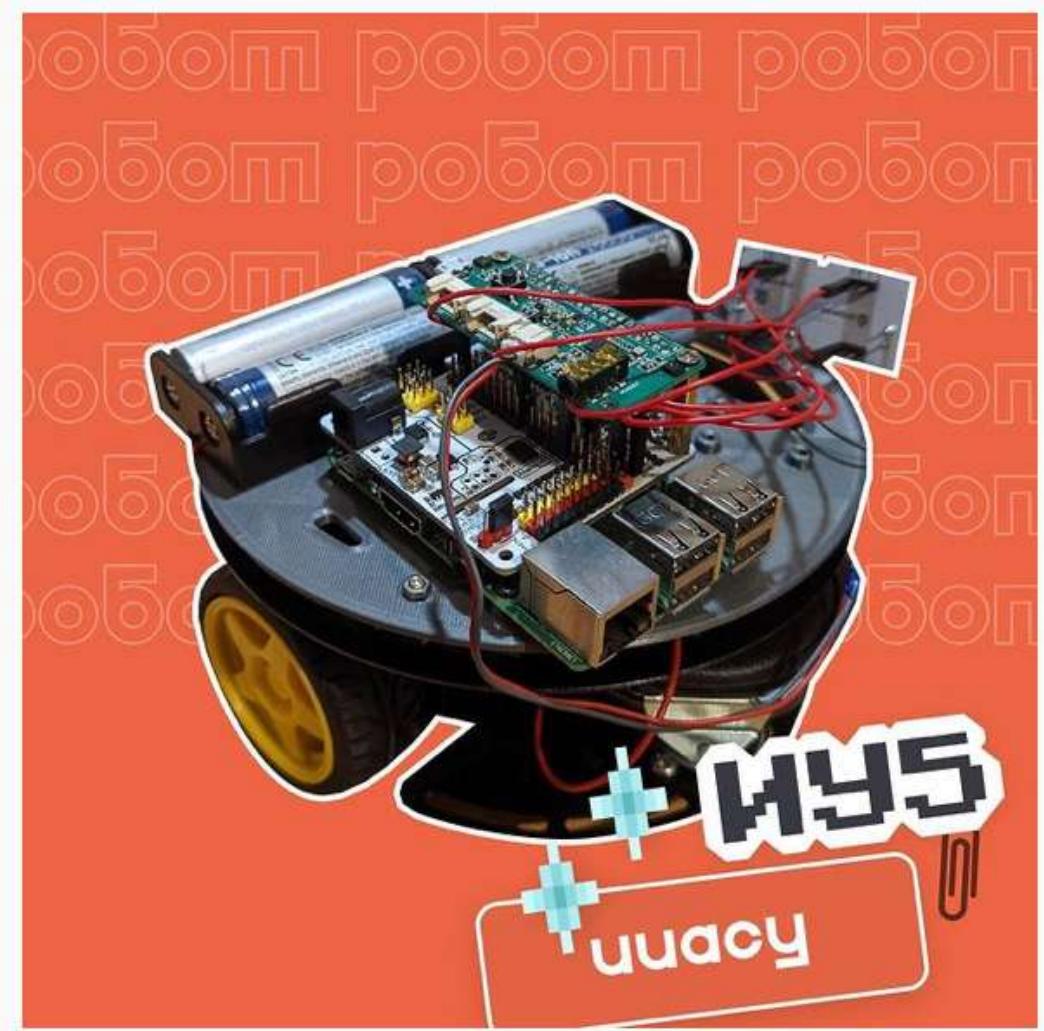
- Оружия
- Масок и тд

Обработка речи и текста

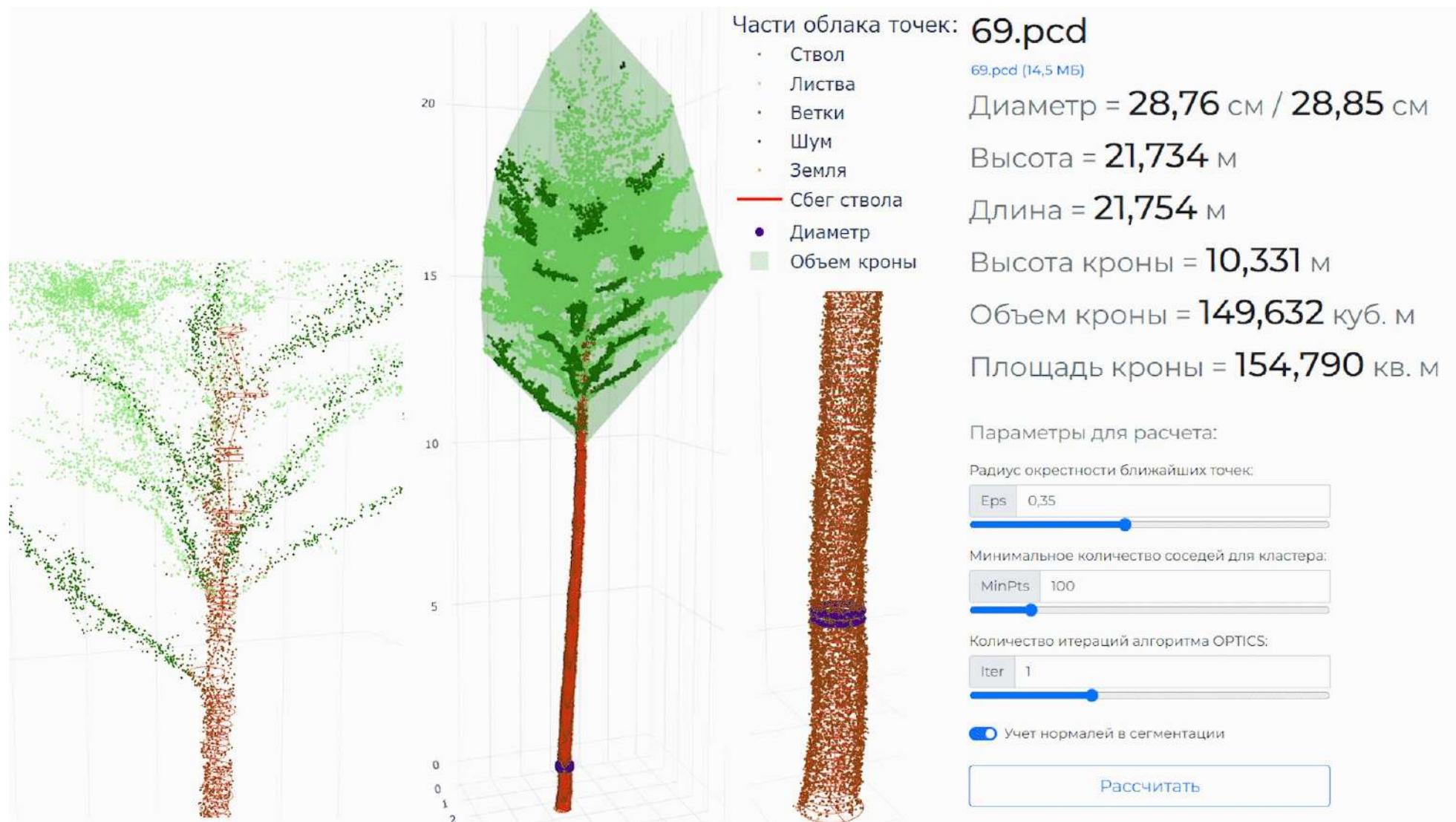
- Автоматическое распознавание речи
- Распознавание сканов - OCR



Areas of investigation



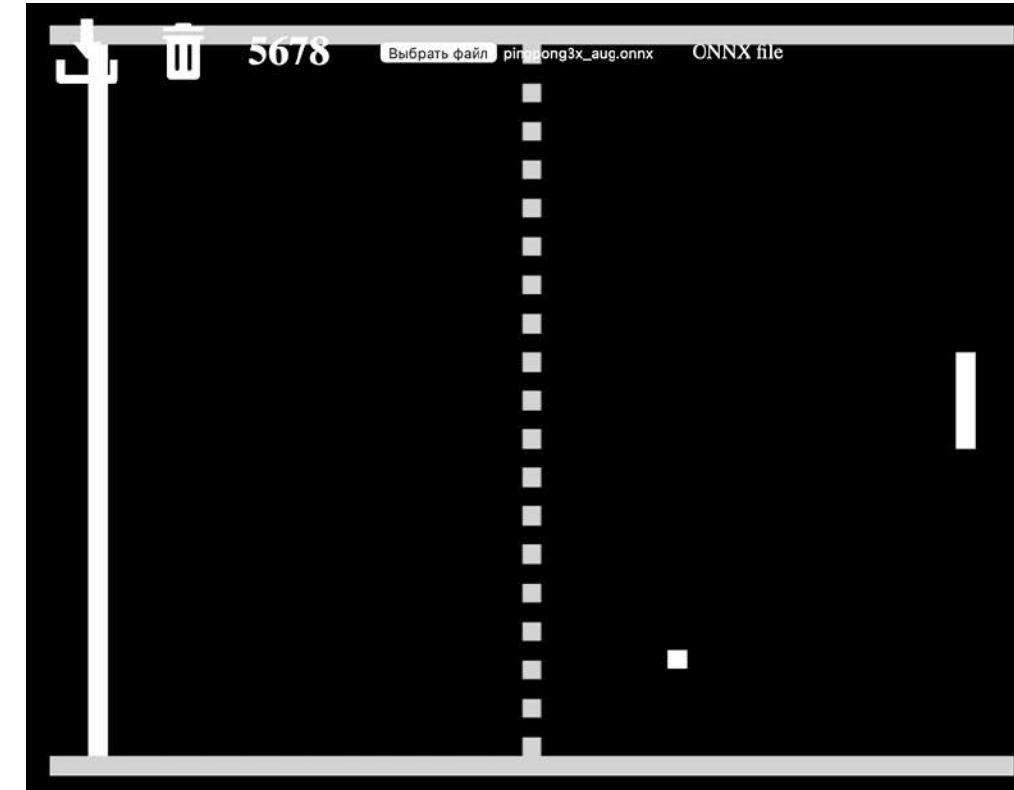
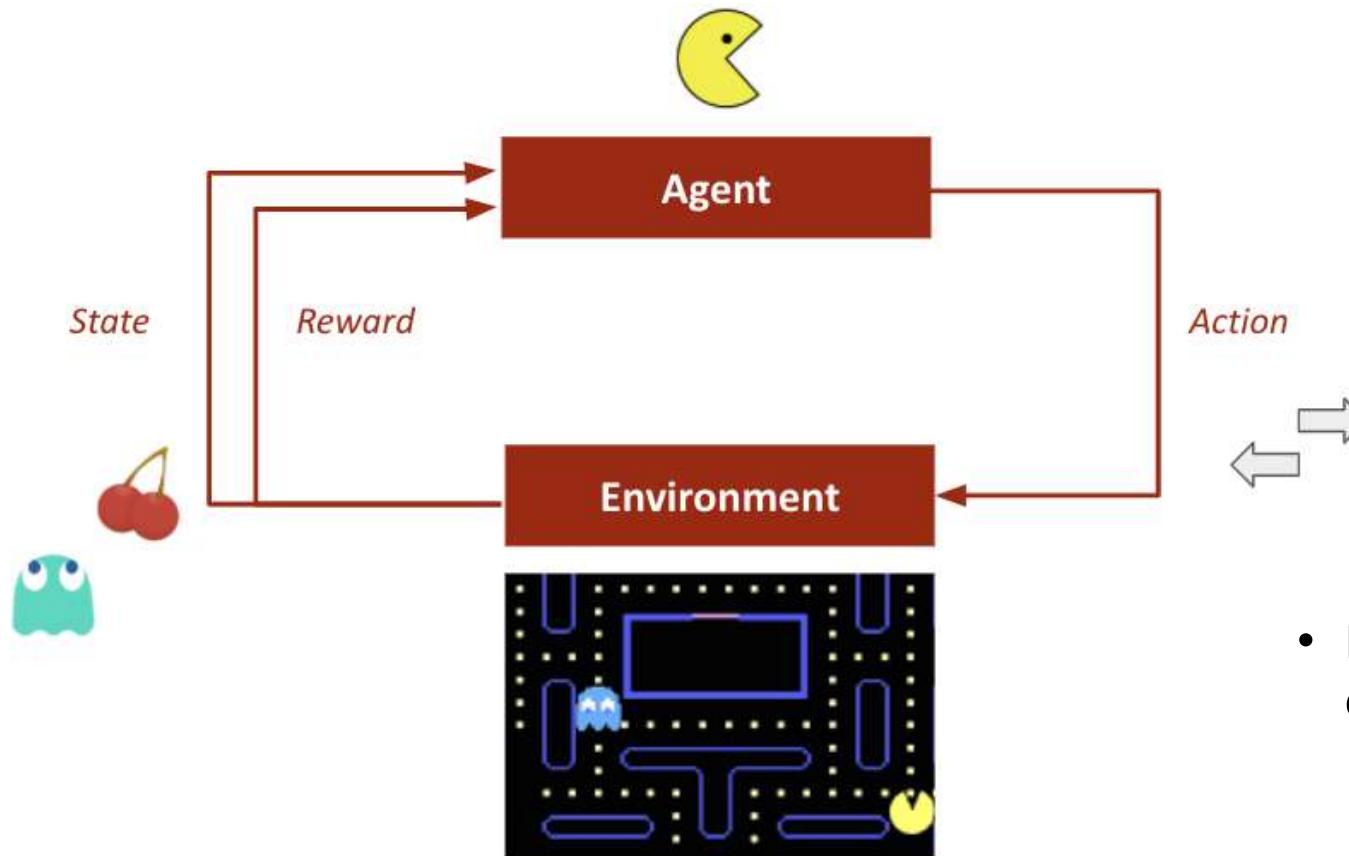
Lidar



Типы машинного обучения

- Обучение с учителем
 - Обучающие данные размечены
 - Задачей является правильно определить класс объекта
- Обучение с подкреплением
 - Обучающие данные не размечены
 - Система получает обратную связь от своих действий
 - Задачей является выбор правильных действий
- Обучение без учителя
 - Обучающие данные не размечены
 - Задачей является правильно определить категорию

Компьютерные игры



- Пример интеллектуального агента ping-pong обученного на кафедре

<https://github.com/iu5git/ai-bot-games-in-js>

- Обучение с подкреплением или по данным игры пользователя

Методы обучения с учителем

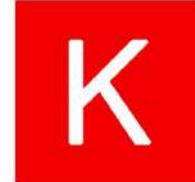
- Линейная регрессия
- Дерево решений
- Машина опорных векторов (SVM)
- К-ближайших соседей
- Нейронные сети

Фреймворки обучения с учителем

 PyTorch



TensorFlow

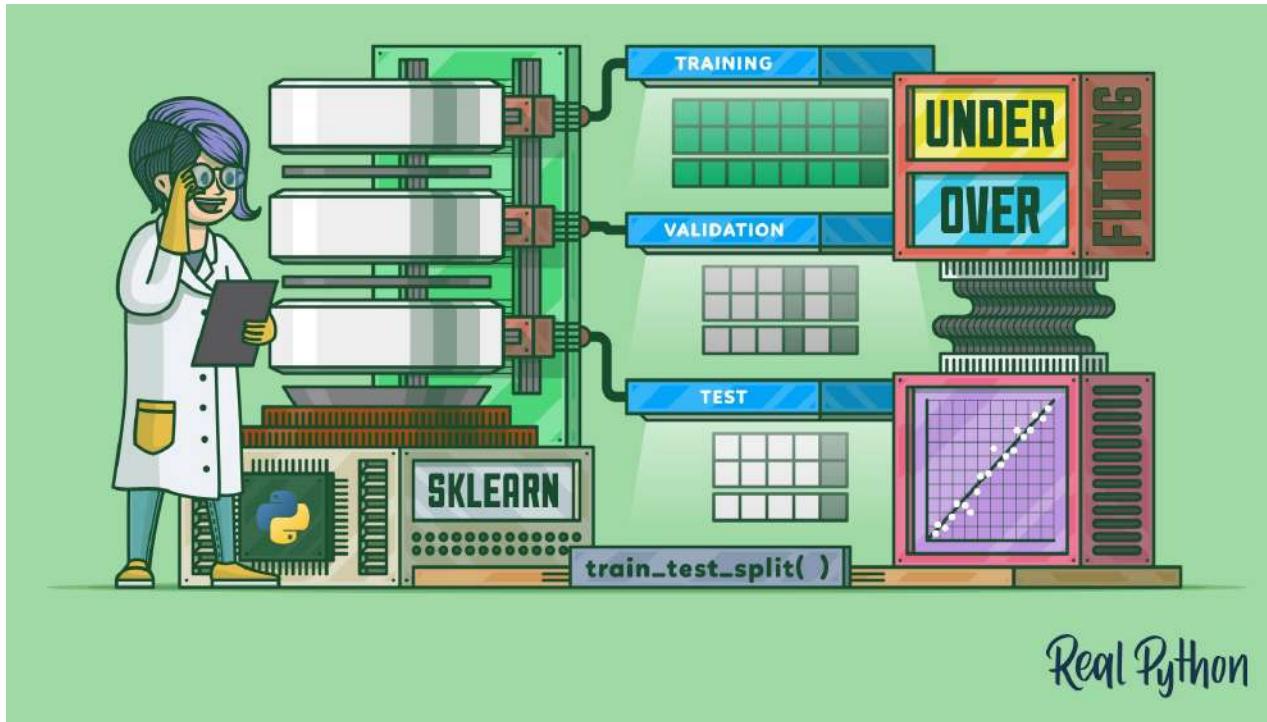
 Keras

Признаки в машинном обучении

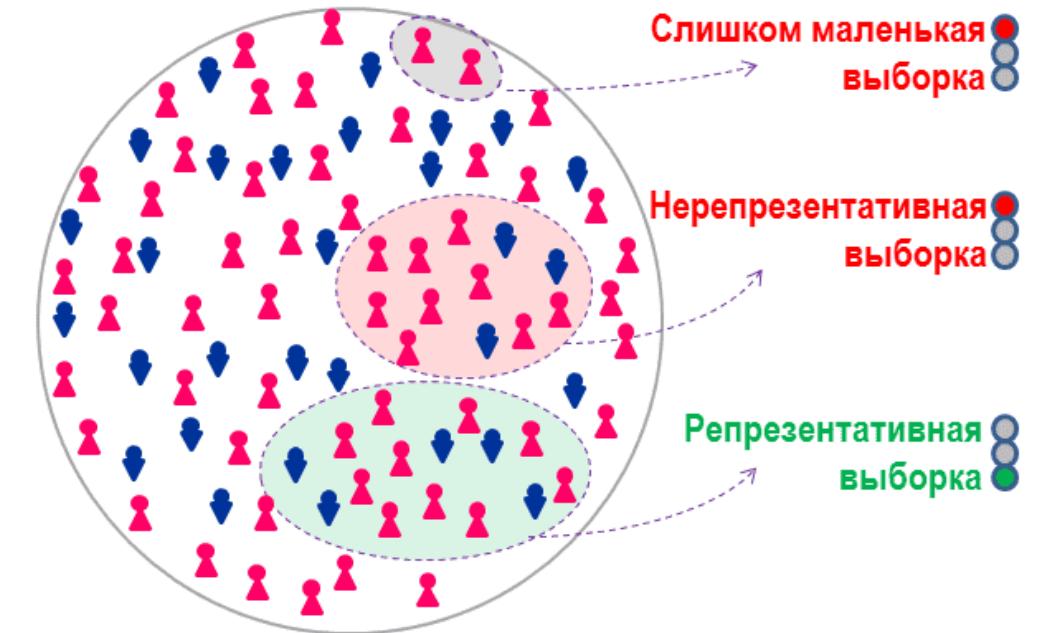
- Признаки – это наблюдения, которые используются для формирования прогнозов
 - Для классификации изображений каждый пиксель является признаком
 - Для распознавания голоса, частота и громкость звуковых примеров являются признаками
 - Для беспилотных автомобилей данные с камер, радаров и GPS являются признаками
- Извлечение подходящих признаков важно для построения модели
 - Время суток - это неподходящий признак при классификации изображений
 - Время суток - это подходящий признак при классификации электронных писем, т.к. SPAM часто приходит по ночам
- Общие типы признаков в робототехнике
 - Пиксели (RGB данные)
 - Глубина (сонар, лазерные дальномеры)
 - Движение (значения с микросхем)
 - Ориентация или ускорение (Гироскоп, Акселерометр, Компас)

Формирование набора данных. Примеры, входные данные и label

- Размер набора данных и распределение по классам
- Части набора данных: входные данные и метки; test и train



Генеральная совокупность включает ♂ - 1/3 и ♀ - 2/3



Тестовая и обучающая выборки

- Входные данные и метки
 - Тестовая и обучающая части набора данных

```
with open('cifar-100-python/train', 'rb') as f:  
    data_train = pickle.load(f, encoding='latin1')  
with open('cifar-100-python/test', 'rb') as f:  
    data_test = pickle.load(f, encoding='latin1')
```

Обучающая выборка

Тестовая выборка

0	0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0
1	1 1 1 1 1 1 1 1 1 1 1 1 1	1 1 1 1 1 1
2	2 2 2 2 2 2 2 2 2 2 2 2 2	2 2 2 2 2 2
3	3 3 3 3 3 3 3 3 3 3 3 3 3	3 3 3 3 3 3
4	4 4 4 4 4 4 4 4 4 4 4 4 4	4 4 4 4 4 4
5	5 5 5 5 5 5 5 5 5 5 5 5 5	5 5 5 5 5 5
6	6 6 6 6 6 6 6 6 6 6 6 6 6	6 6 6 6 6 6
7	7 7 7 7 7 7 7 7 7 7 7 7 7	7 7 7 7 7 7
8	8 8 8 8 8 8 8 8 8 8 8 8 8	8 8 8 8 8 8
9	9 9 9 9 9 9 9 9 9 9 9 9 9	9 9 9 9 9 9

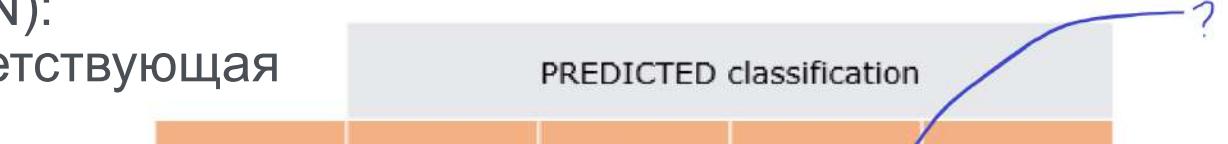
Пример: Определение кошек

Прогноз:						
Картина:						
Истинно Положительное (TP)	Истинно Отрицательное (TN)	Ложно Отрицательное (FN)	Ложно Положительное (FP)			

Матрица ошибок

- Истинно положительные (**True Positive**, TP): Правильно определенная как соответствующая
- Истинно отрицательные (**True Negative**, TN): Правильно определенная как не соответствующая
- Ложно положительные (**False Positive**, FP): Неправильно определенная как соответствующая
- Ложно отрицательные (**False Negative**, FN): Неправильно определенная как не соответствующая
- Confusion Matrix для нескольких классов

		PREDICTED classification			
		a	b	c	d
ACTUAL classification	a	TN	FP	TN	TN
	b	FN	TP	FN	FN
	c	TN	FP	TN	TN
	d	TN	FP	TN	TN



Метрики для классификации

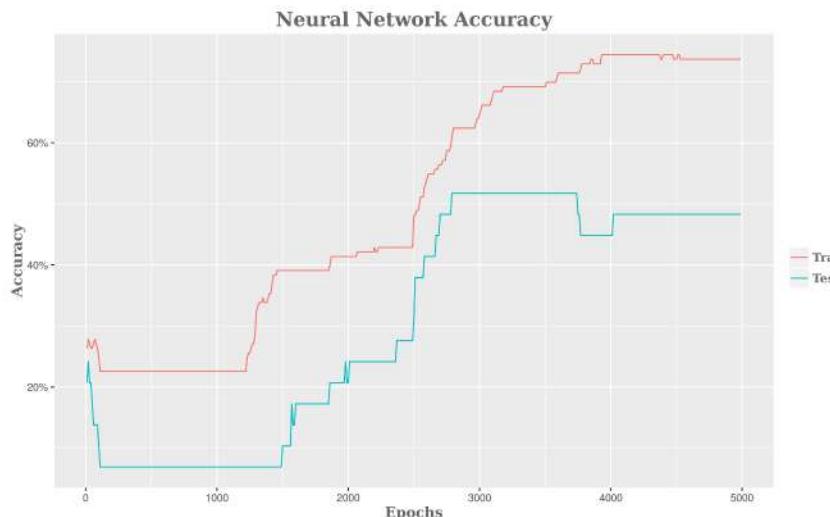
- Точность (Precision)
 - Процент положительных меток которые правильно определены
 - $Precision = (\# \text{ true positives}) / (\# \text{ true positives} + \# \text{ false positives})$
- Полнота (Recall)
 - Процент положительных примеров которые были правильно определены
 - $Recall = (\# \text{ true positives}) / (\# \text{ true positives} + \# \text{ false negatives})$
- Accuracy
 - Процент положительных меток
 - $Accuracy = (\# \text{ true positives} + \# \text{ true negatives}) / (\# \text{ of samples})$

Остальные метрики

https://en.m.wikipedia.org/wiki/Sensitivity_and_specificity

Обучающие и тестовые данные

- Обучающие данные
 - Данные, на которых проводится обучение модели
- Тестовые данные
 - Данные, на которых проводится измерение точности модели
- Переобучение (overfitting)
 - Модель которая хорошо работает на обучающих данных и плохо на тестовых данных



Смещение и разброс (Bias and variance)

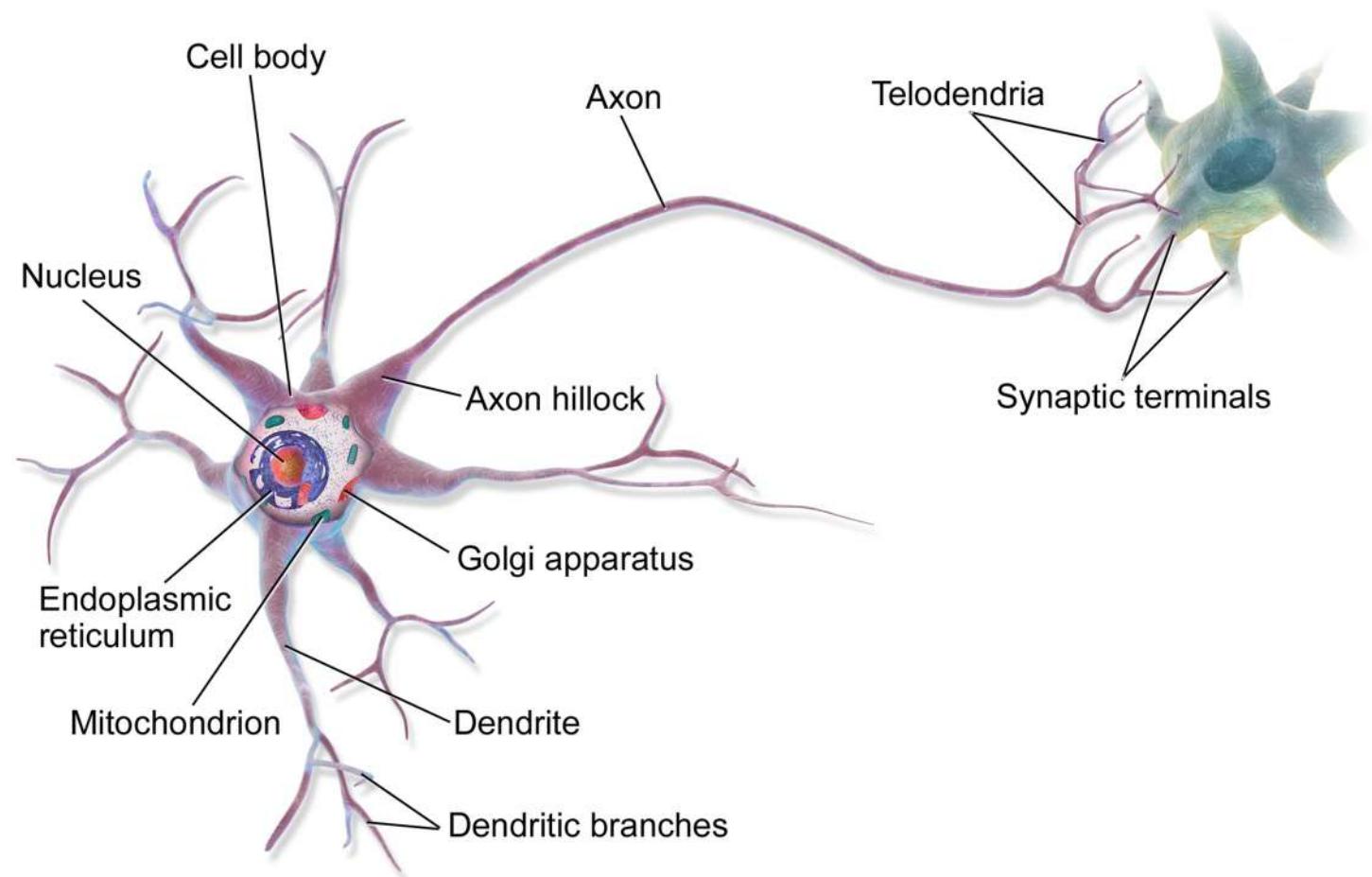
- Смещение: ожидаемая разница между прогнозами модели и истиной
- Разброс: Как сильно отличается ответ модели в обучающих наборах данных
- Возможные сценарии модели
 - Большое смещение: модель делает неправильный прогноз на обучающих примерах
 - Большой разброс: Модель не обобщает на новых наборах данных
 - Маленькое смещение: Модель делает правильный прогноз на обучающих данных
 - Маленький разброс: Модель обобщает на новых наборах данных

Результаты из лабораторной

train		precision	recall	f1-score	support
	0	0.9940	1.0000	0.9970	500
	55	1.0000	0.9900	0.9950	500
	58	0.9960	1.0000	0.9980	500
		accuracy		0.9967	1500
macro avg		0.9967	0.9967	0.9967	1500
weighted avg		0.9967	0.9967	0.9967	1500

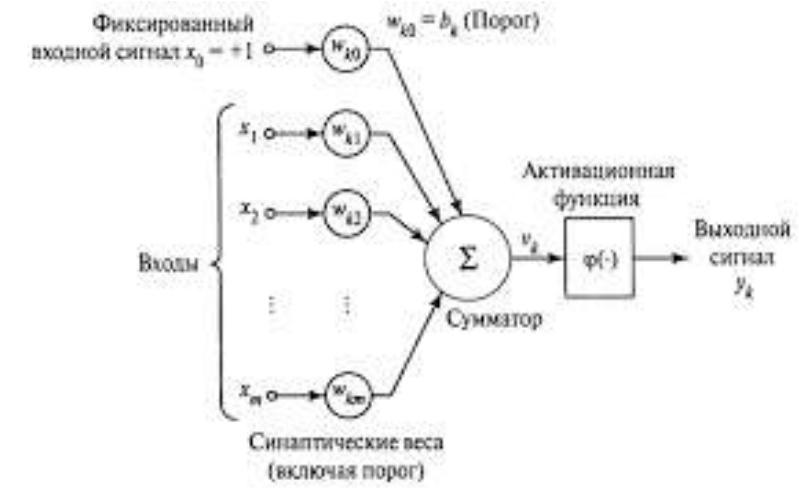
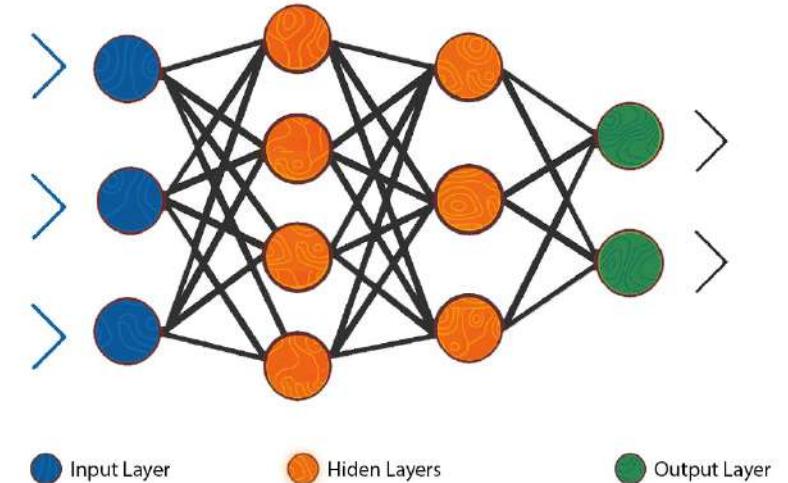
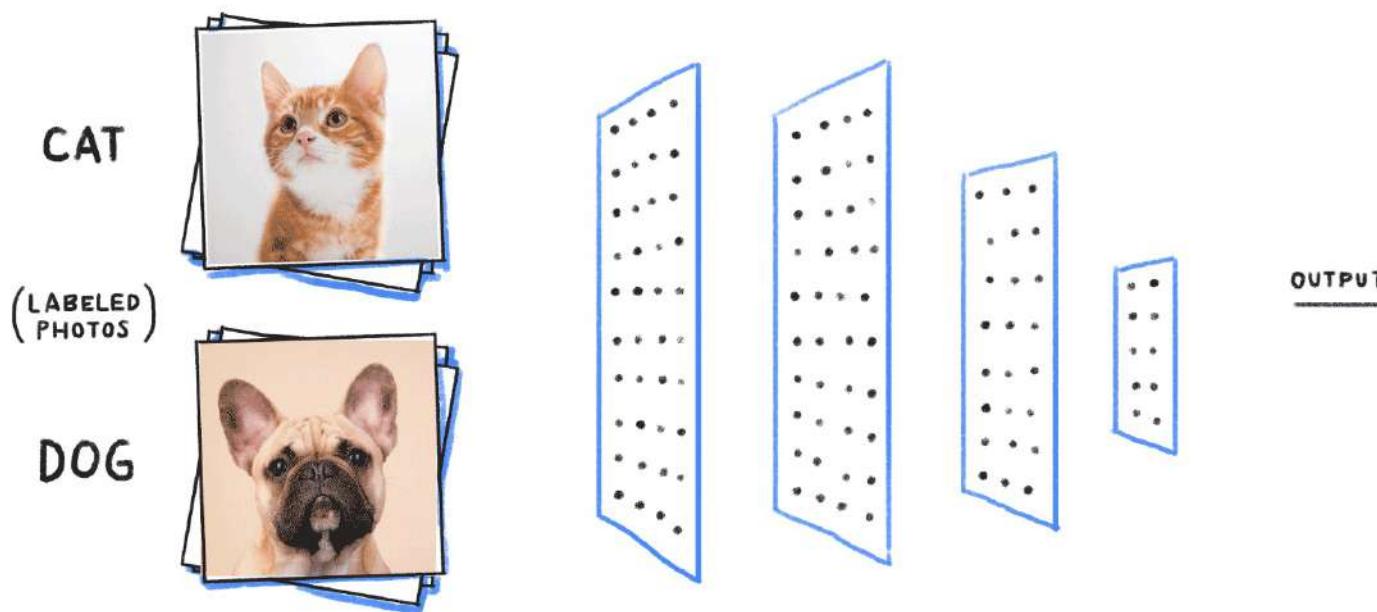
test		precision	recall	f1-score	support
	0	0.8120	0.9500	0.8756	100
	55	0.7396	0.7100	0.7245	100
	58	0.7471	0.6500	0.6952	100
		accuracy		0.7700	300
macro avg		0.7662	0.7700	0.7651	300
weighted avg		0.7662	0.7700	0.7651	300

Биологическая ассоциация



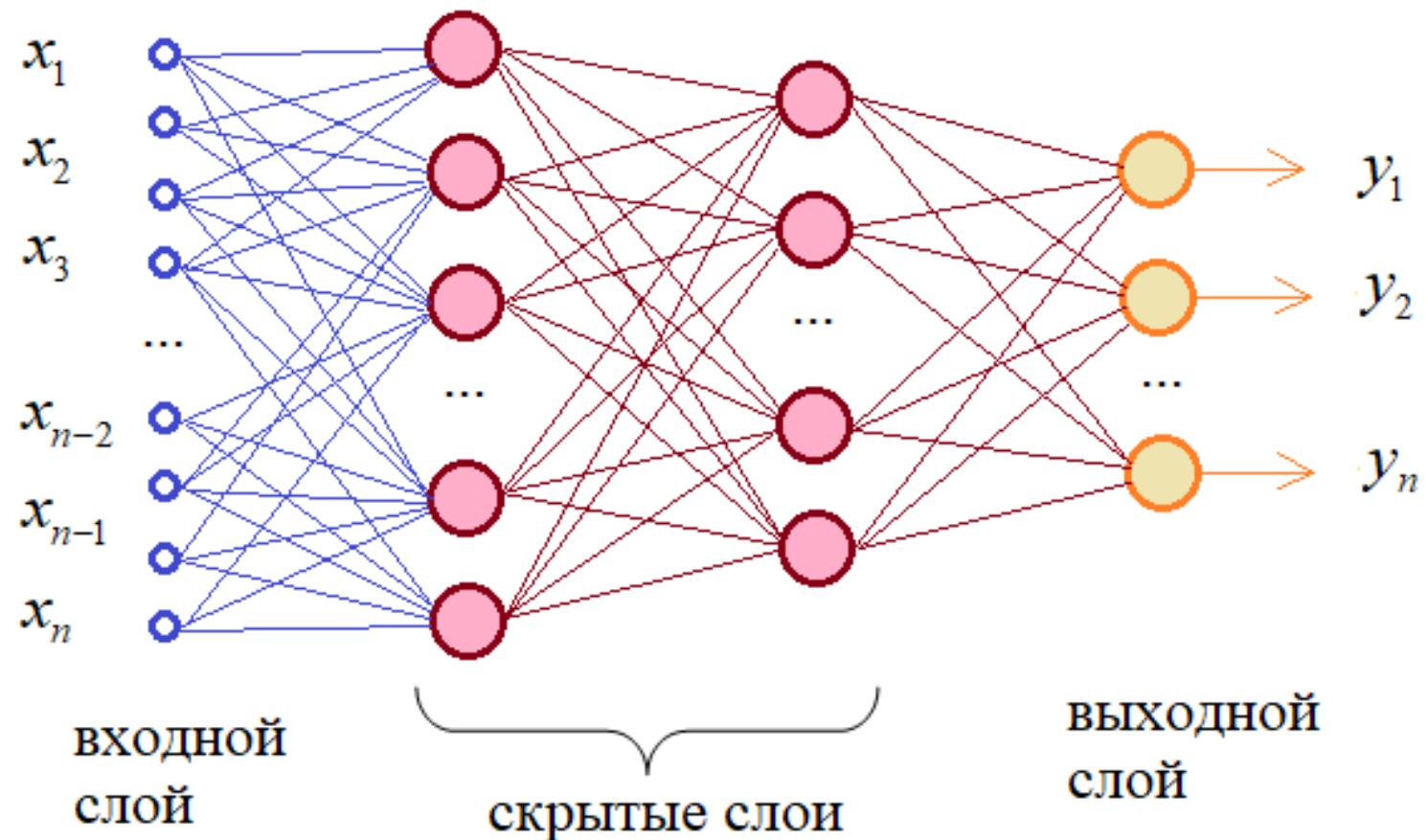
Нейронная сеть

- Сеть состоит из слоев нейронов
- Каждый нейрон – сумматор
- Обучение – вычисление весов w нейрона



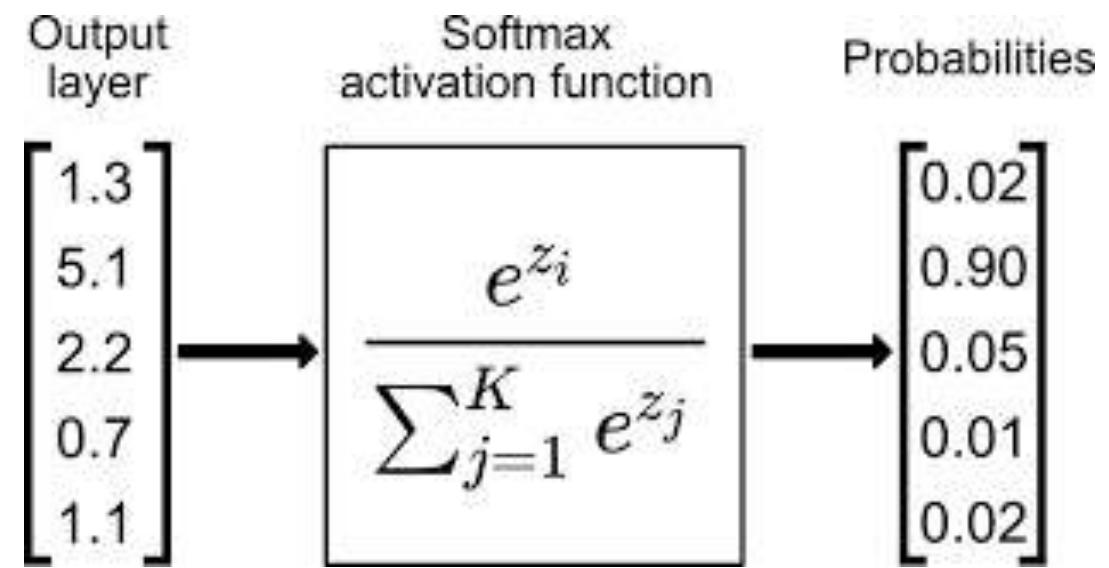
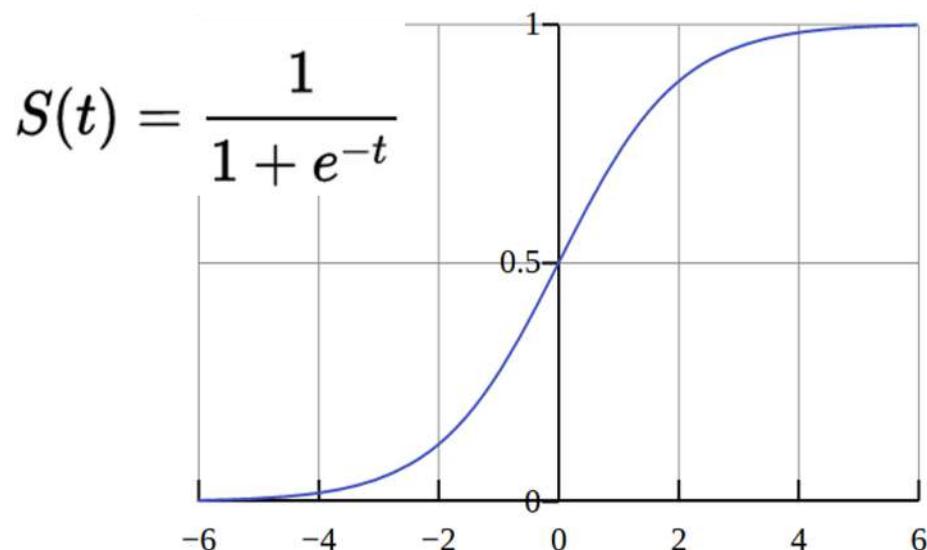
Полносвязная нейросеть

- Хорошая классификация при малом объеме входных данных
- Очень много связей при большом объеме данных



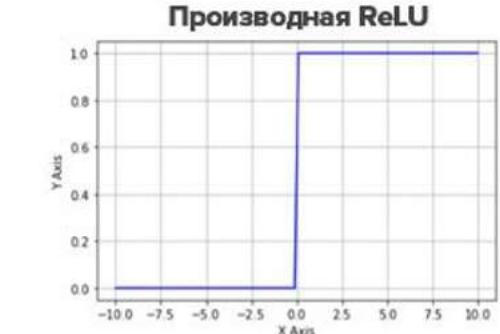
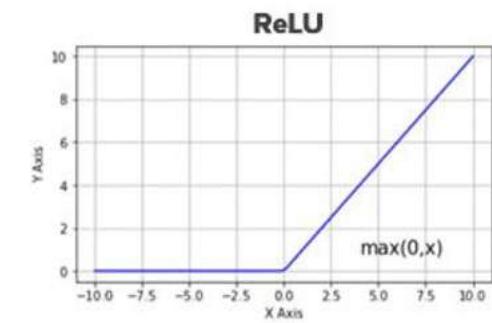
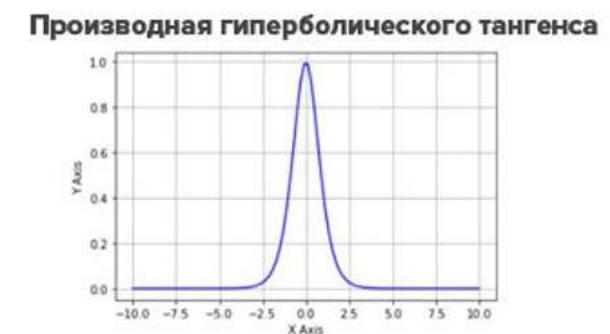
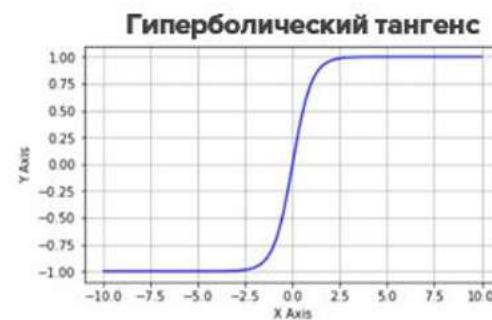
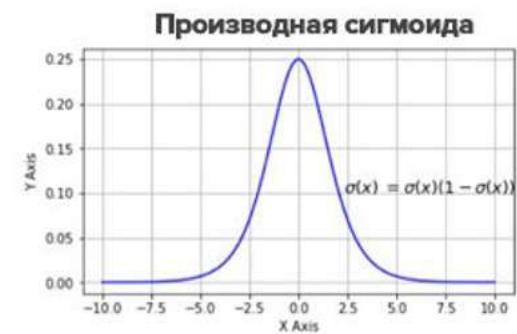
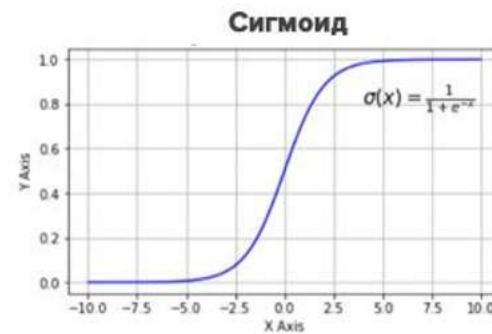
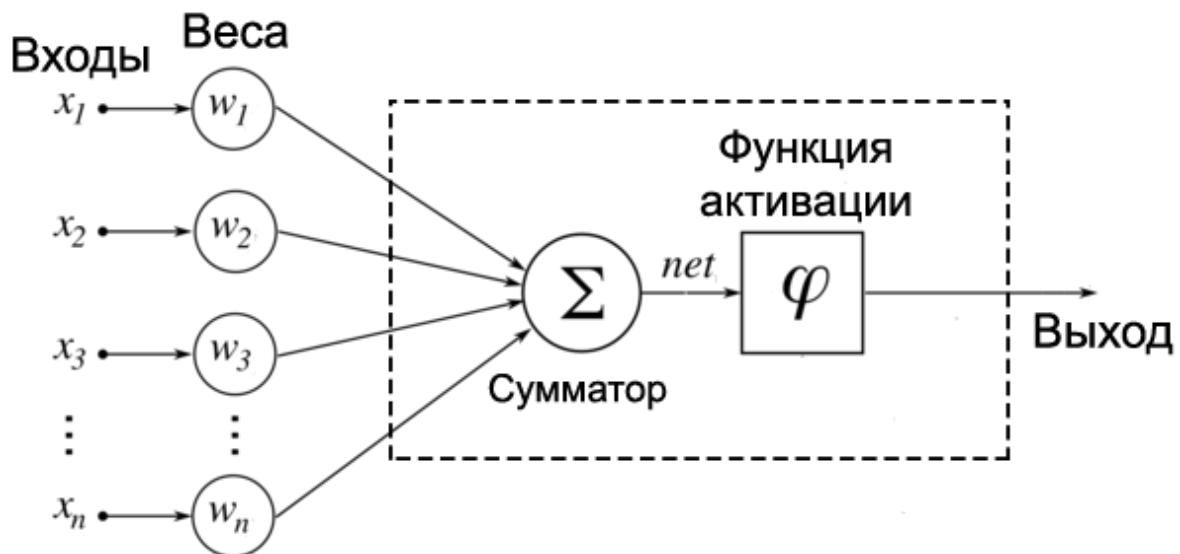
Активационные функции

- Активационные функции применяются ко всем входным значениям в каждом нейроне
 - Сигмоидальная функция является распространенной активационной функцией
 - Softmax – это вариант сигмоиды для нескольких классов

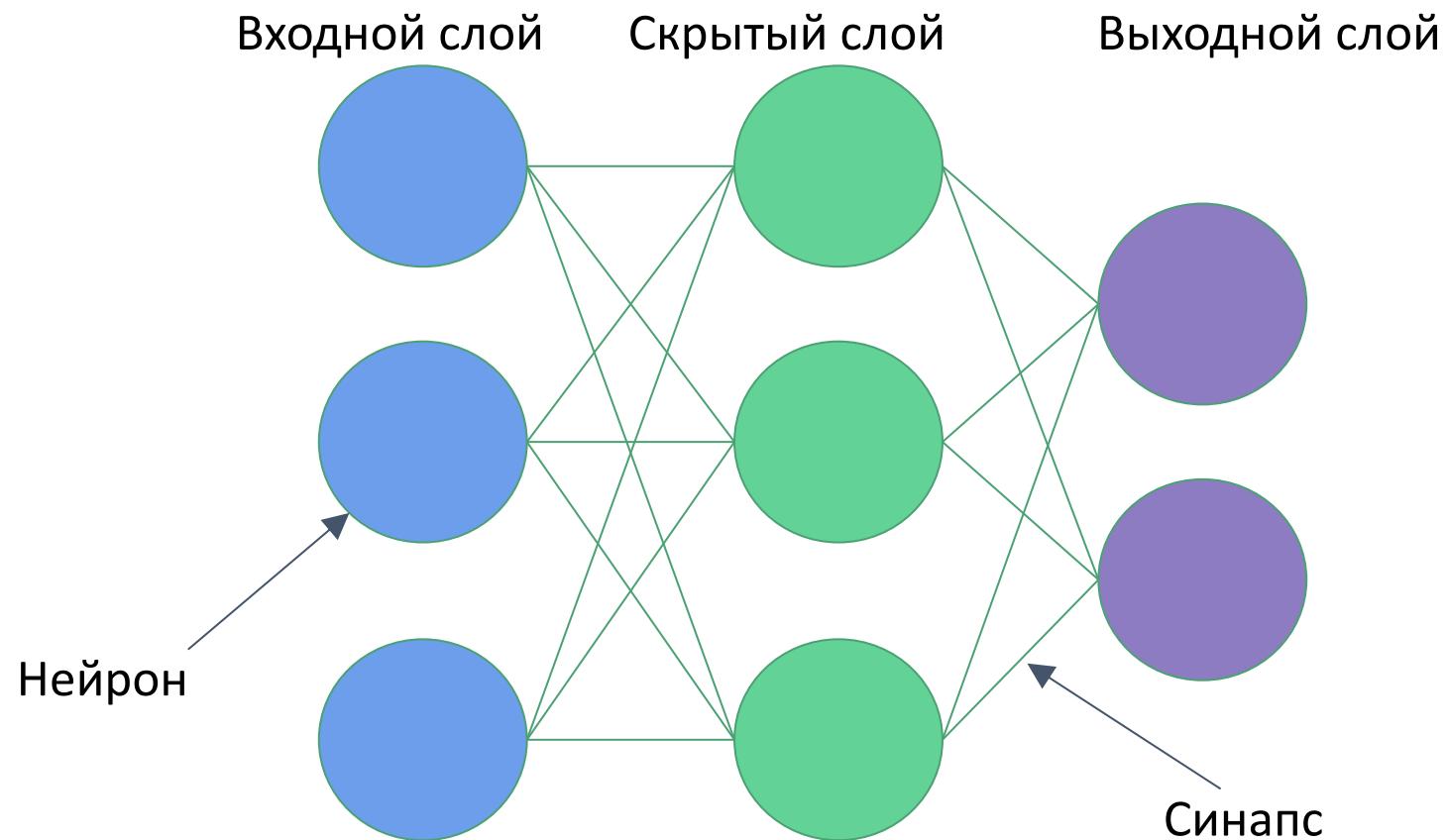


Активационная функция

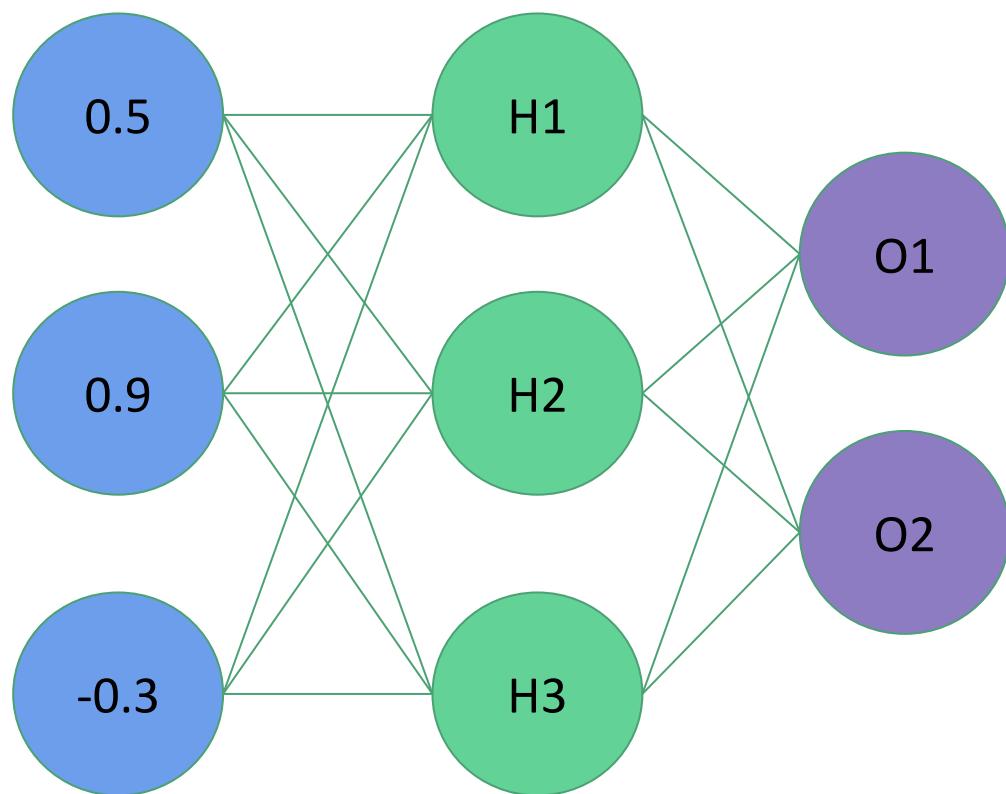
- Помимо сумматора есть активационная функция
- Они разные для разных задач



Архитектура нейронной сети



Вычисление прогноза (инференс)



Веса H1 = (1.0, -2.0, 2.0)

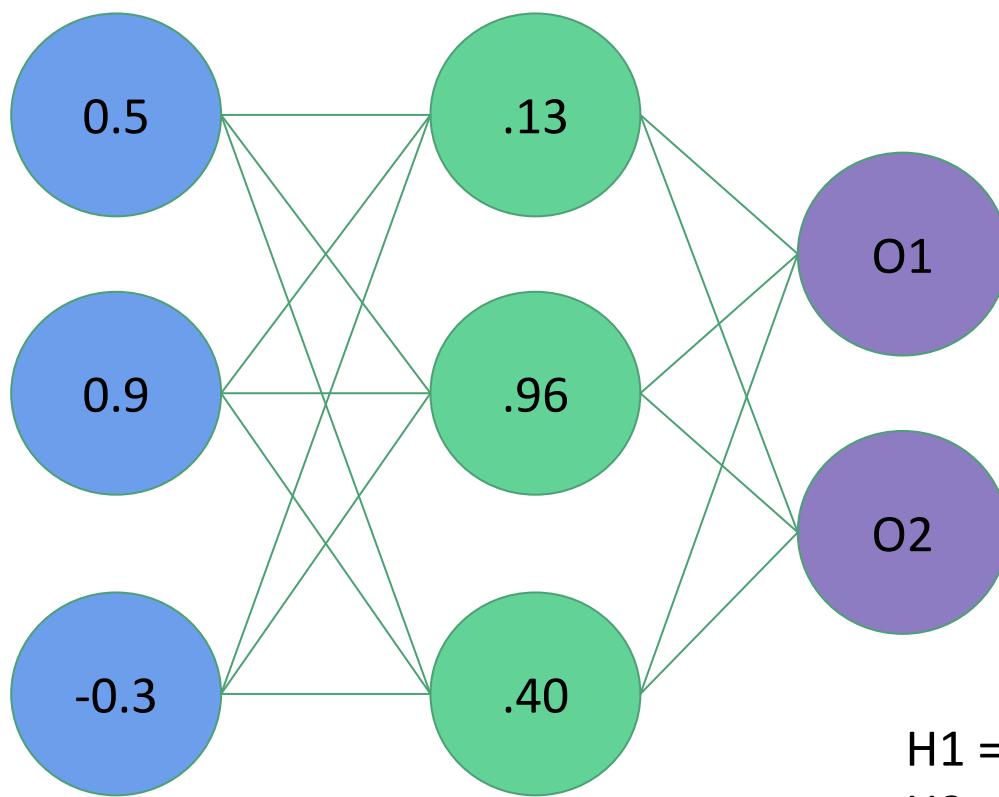
Веса H2 = (2.0, 1.0, -4.0)

Веса H3 = (1.0, -1.0, 0.0)

Веса O1 = (-3.0, 1.0, -3.0)

Веса O2 = (0.0, 1.0, 2.0)

Вычисление прогноза (инференс)



$$\text{Веса } H1 = (1.0, -2.0, 2.0)$$

$$\text{Веса } H2 = (2.0, 1.0, -4.0)$$

$$\text{Веса } H3 = (1.0, -1.0, 0.0)$$

$$\text{Веса } O1 = (-3.0, 1.0, -3.0)$$

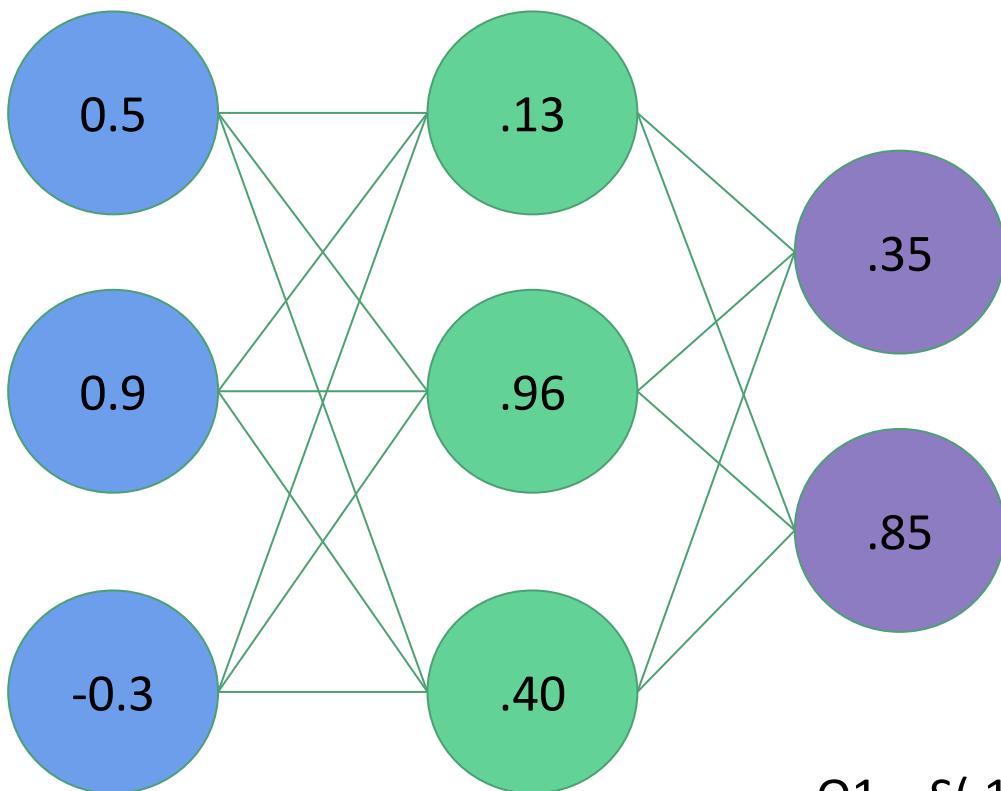
$$\text{Веса } O2 = (0.0, 1.0, 2.0)$$

$$H1 = S(0.5 * 1.0 + 0.9 * -2.0 + -0.3 * 2.0) = S(-1.9) = .13$$

$$H2 = S(0.5 * 2.0 + 0.9 * 1.0 + -0.3 * -4.0) = S(3.1) = .96$$

$$H3 = S(0.5 * 1.0 + 0.9 * -1.0 + -0.3 * 0.0) = S(-0.4) = .40$$

Вычисление прогноза (инференс)



$$\text{Веса } H1 = (1.0, -2.0, 2.0)$$

$$\text{Веса } H2 = (2.0, 1.0, -4.0)$$

$$\text{Веса } H3 = (1.0, -1.0, 0.0)$$

$$\text{Веса } O1 = (-3.0, 1.0, -3.0)$$

$$\text{Веса } O2 = (0.0, 1.0, 2.0)$$

$$O1 = S(0.13 * -3.0 + 0.96 * 1.0 + 0.40 * -3.0) = S(-0.63) = 0.35$$

$$O2 = S(0.13 * 0.0 + 0.96 * 1.0 + 0.40 * 2.0) = S(1.76) = 0.85$$

Матричное представление

Веса H1 = (1.0, -2.0, 2.0)

Веса H2 = (2.0, 1.0, -4.0)

Веса H3 = (1.0, -1.0, 0.0)

Веса скрытого слоя Вход

$S($

1.0	-2.0	2.0
2.0	1.0	-4.0
1.0	-1.0	0.0

 *

0.5
0.9
-0.3

) = $S($

-1.9	3.1	-0.4
------	-----	------

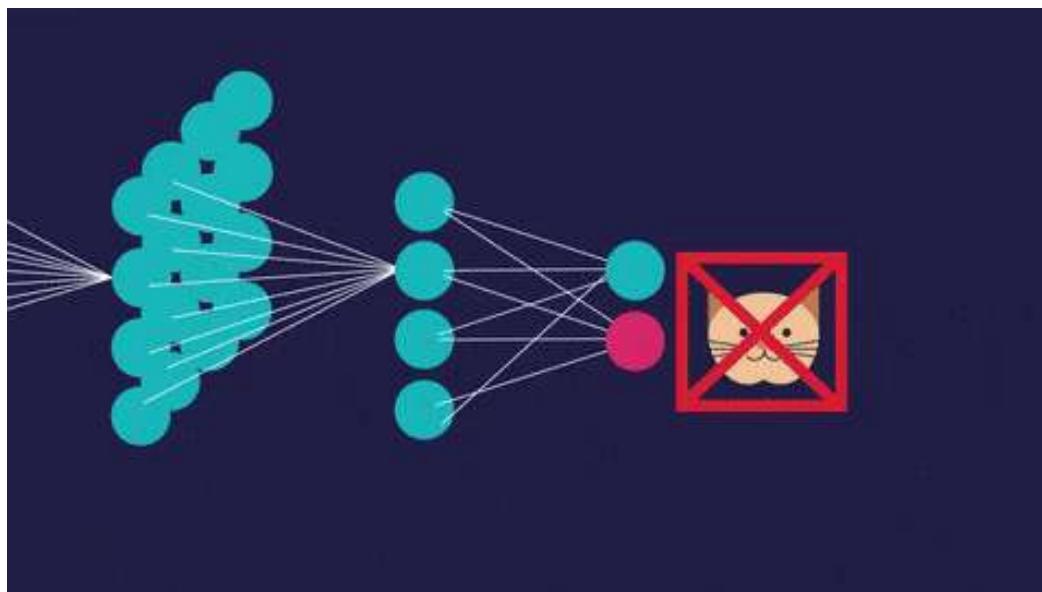
) =

.13	.96	0.4
-----	-----	-----

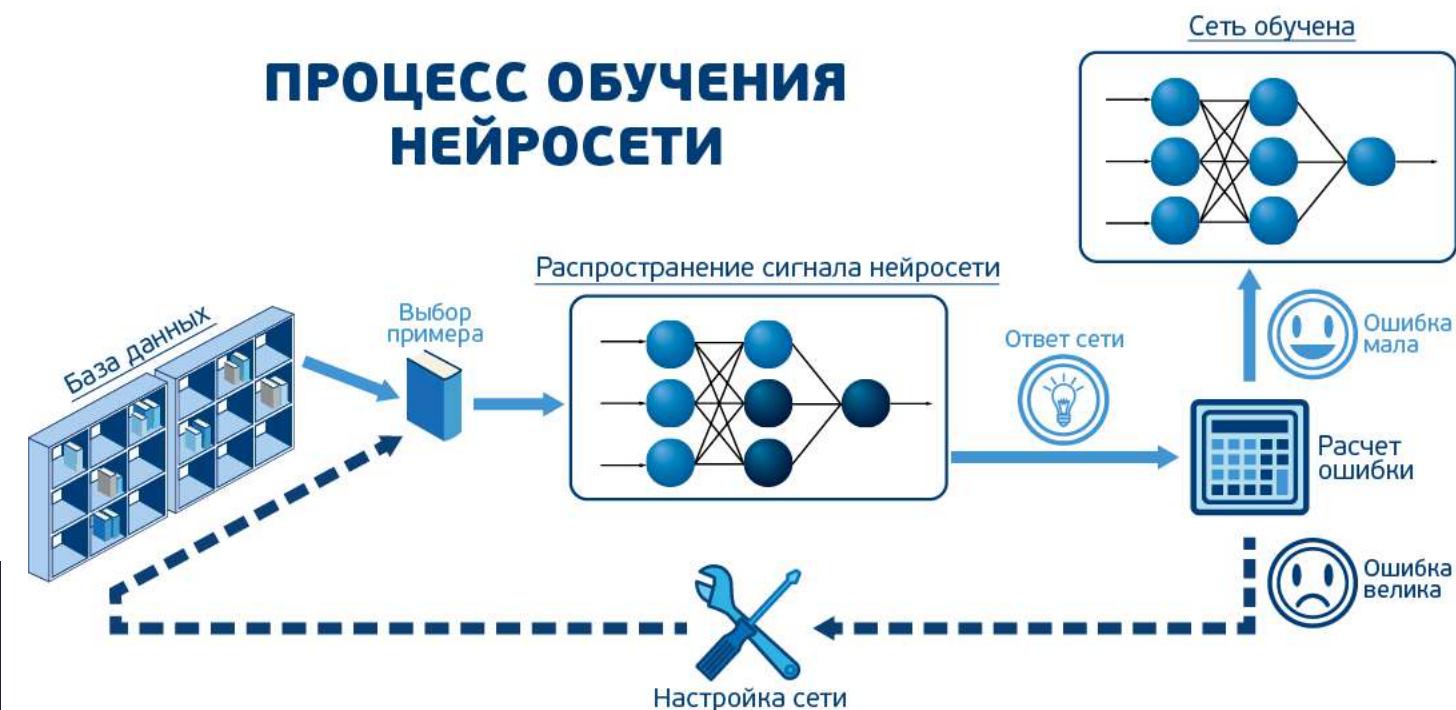
 Значения скрытого слоя

Обучение нейросети

- Обучение нейросети
- Обучение с учителем



ПРОЦЕСС ОБУЧЕНИЯ НЕЙРОСЕТИ



Обучение нейронных сетей

- Процедуры для обучения нейронных сетей
 - Произвести инференс на обучающем наборе
 - Вычислить ошибку между спрогнозированными значениями и истинными значениями на обучающем наборе
 - Определите вклад каждого нейрона в ошибку
 - Изменить веса нейронной сети для минимизации ошибки
- Вклад в размер ошибки вычисляется с помощью обратного распространения (Backpropagation)
- Минимизация ошибки достигается благодаря градиентному спуску (Gradient Descent)

Метод градиентного спуска

- Метод градиентного спуска минимизирует ошибку нейронной сети
 - На каждой итерации ошибка сети рассчитывается на основе обучающих данных
 - Затем модифицируются веса для уменьшения ошибки
- Метод градиентного спуска останавливается когда:
 - Ошибка достаточно мала
 - Максимальное количество итераций превышено

Обучение

```
EPOCHS = 250
steps_per_epoch = len(dataloader['train'])
steps_per_epoch_val = len(dataloader['test'])
for epoch in range(EPOCHS): # проход по набору данных несколько раз
    running_loss = 0.0
    model.train()
    for i, batch in enumerate(dataloader['train'], 0):
        # получение одного минибатча; batch это двухэлементный список из [inputs, labels]
        inputs, labels = batch

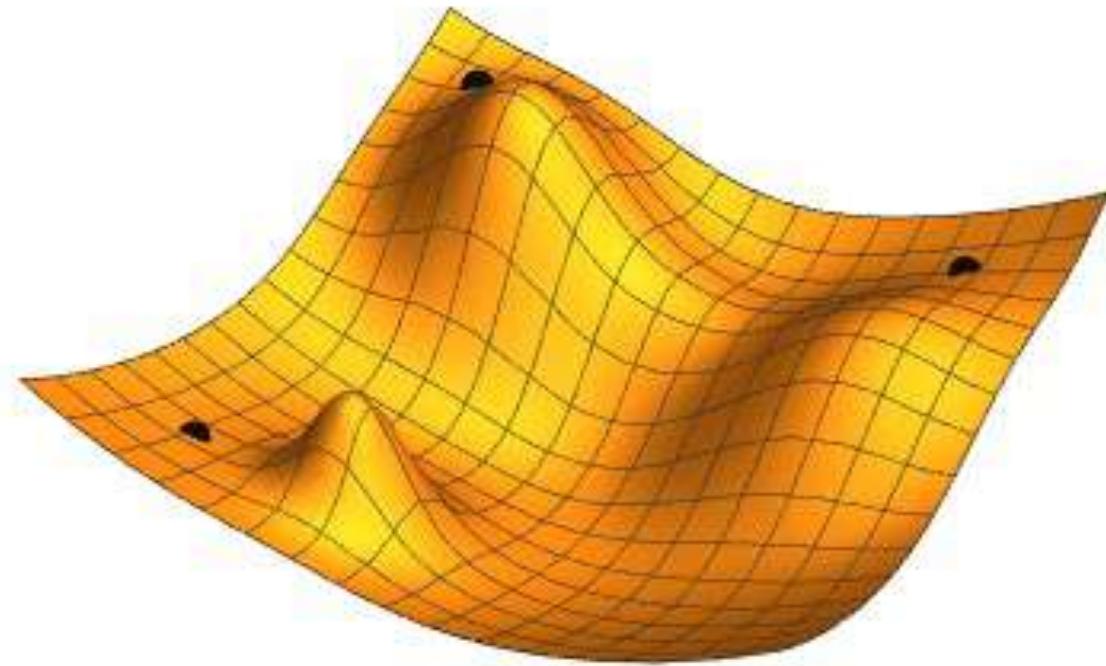
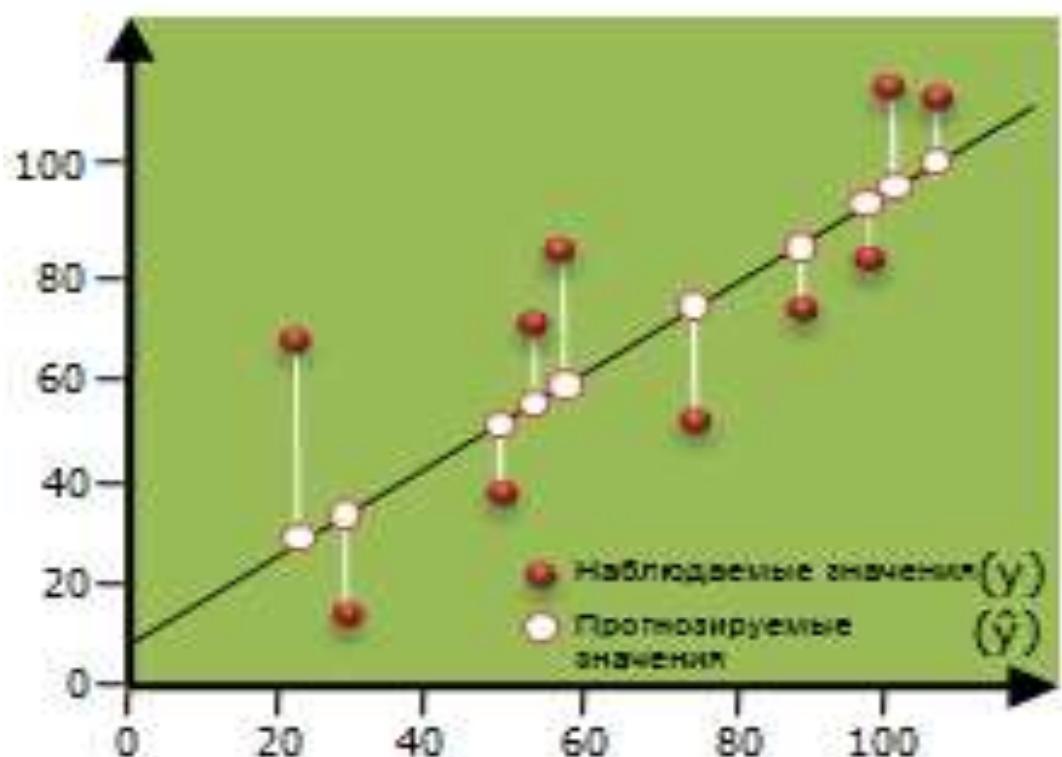
        # очищение прошлых градиентов с прошлой итерации
        optimizer.zero_grad()

        # прямой + обратный проходы + оптимизация
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        #loss = F.cross_entropy(outputs, labels)
        loss.backward()

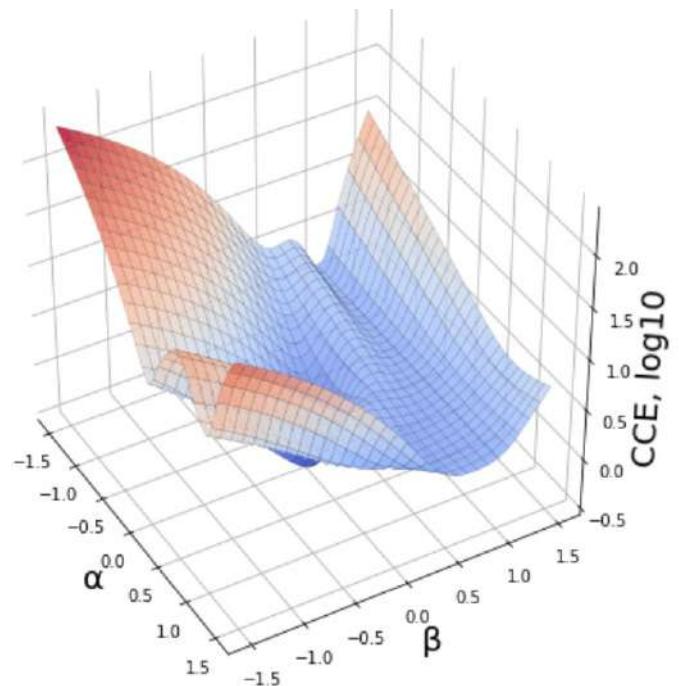
        #Для обновления параметров нейронной сети используется метод step, применённый к экземпляру
        optimizer.step()
```

ФУНКЦИЯ ПОТЕРЬ (loss)

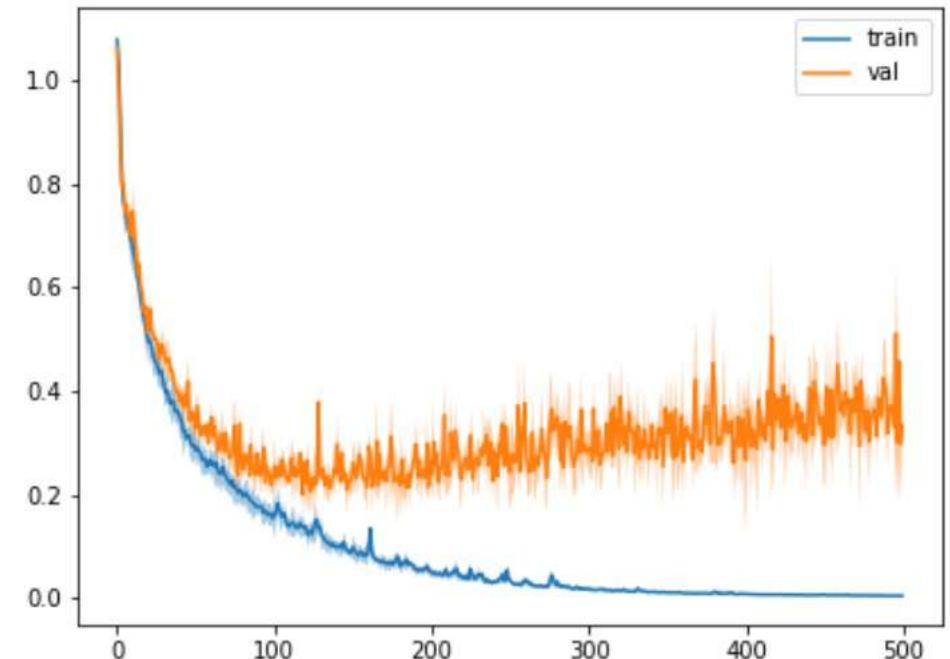
- Нам нужен минимум функции потерь
- Метод наименьших квадратов
- Кроссэнтропия



ФУНКЦИИ ПОТЕРЬ



Примеры из лабораторной



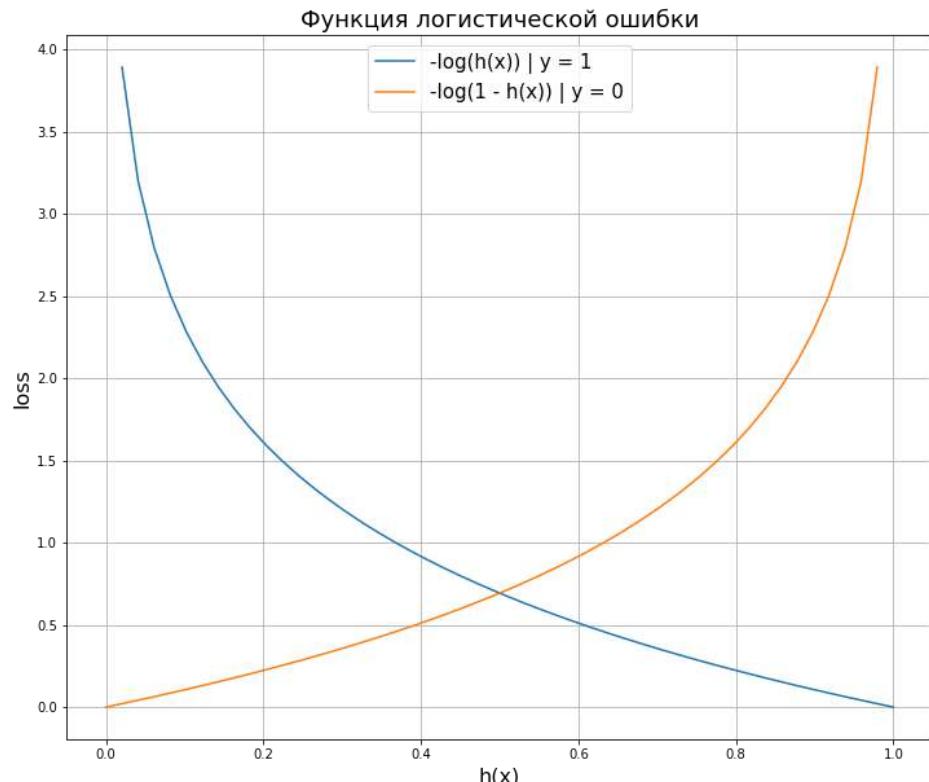
```
criterion = nn.CrossEntropyLoss()
```

```
# Logloss
```

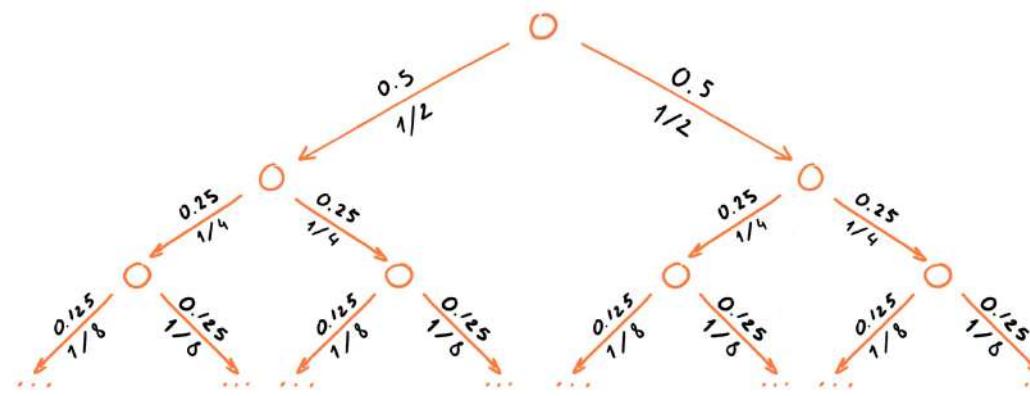
```
def loss(y_true, y_pred):  
    return -1*(y_true*torch.log(y_pred)+(1-y_true)*torch.log(1-y_pred)).sum()
```

Перекрестная энтропия

- В задачах классификации в качестве функции потерь используется перекрестная энтропия – cross entropy
- Случай бинарной кросс-энтропии



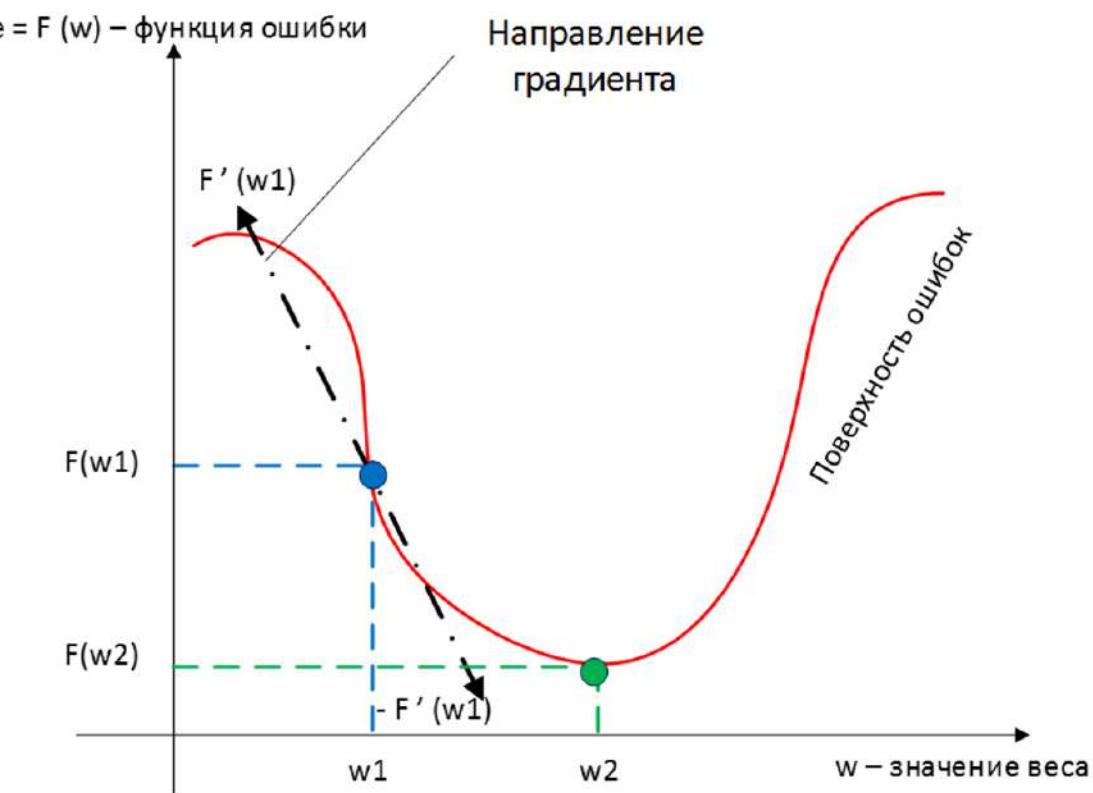
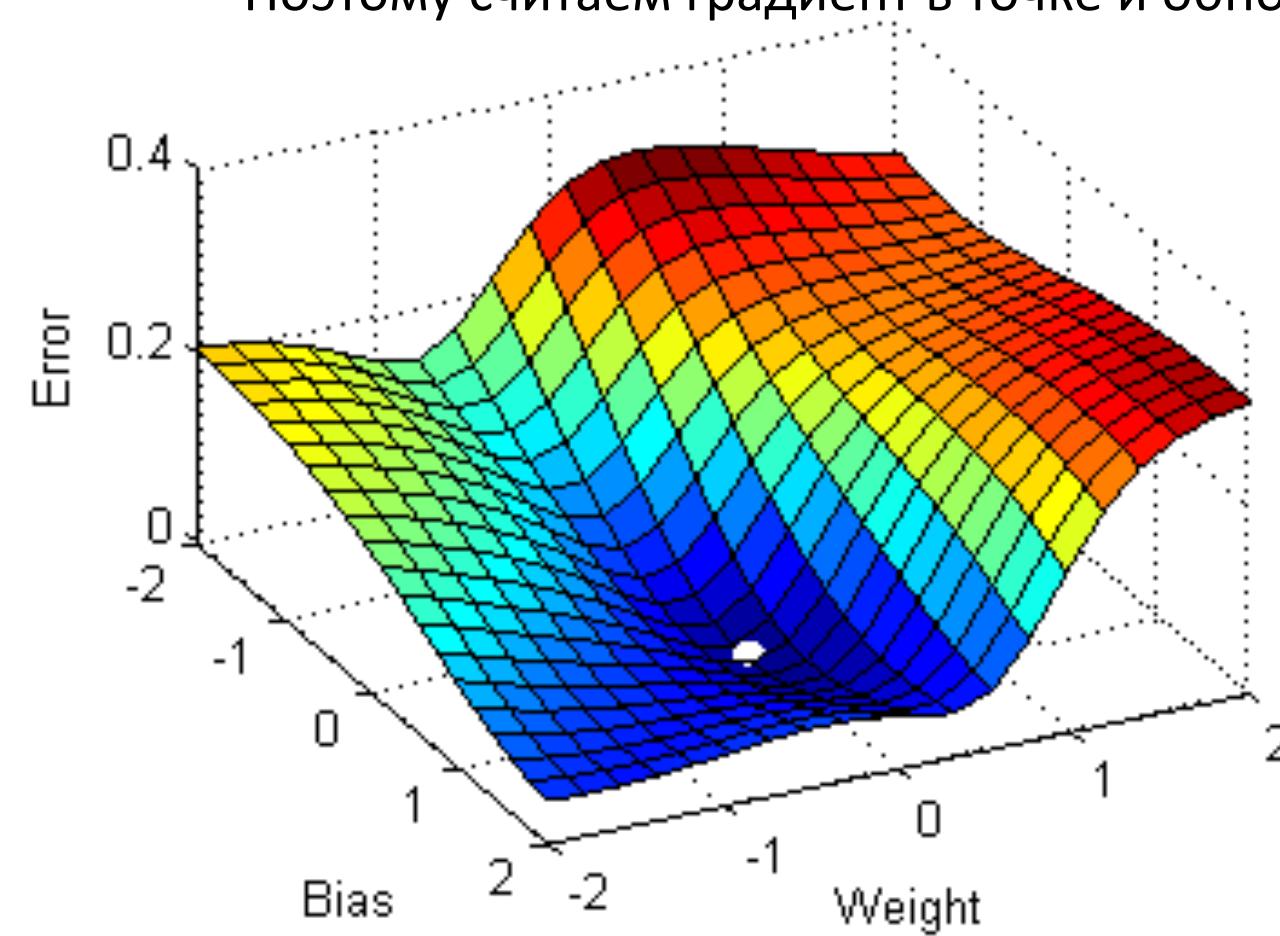
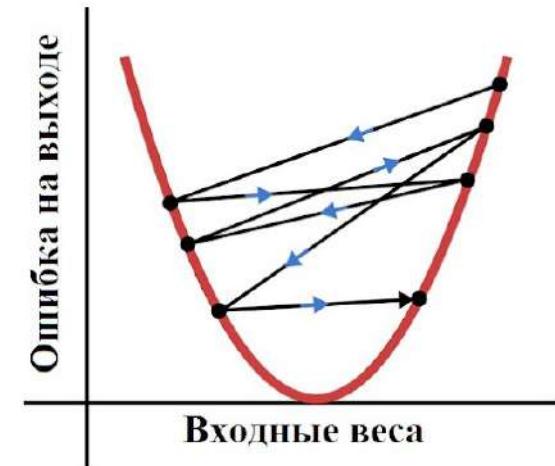
$$H_p(q) = -\frac{1}{N} \sum_{i=1}^N y_i \cdot \log(p(y_i)) + (1 - y_i) \cdot \log(1 - p(y_i))$$



$2^0 = 1$	$\log_2(1) = 0$
$2^1 = 2$	$\log_2(2) = 1$
$2^2 = 4$	$\log_2(4) = 2$
$2^3 = 8$	$\log_2(8) = 3$

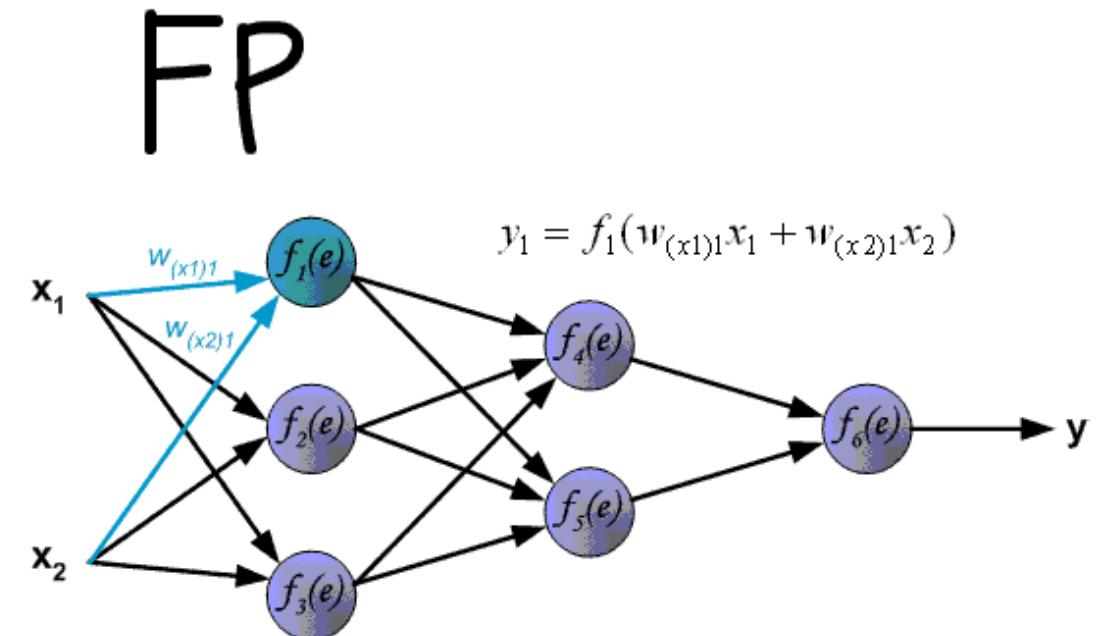
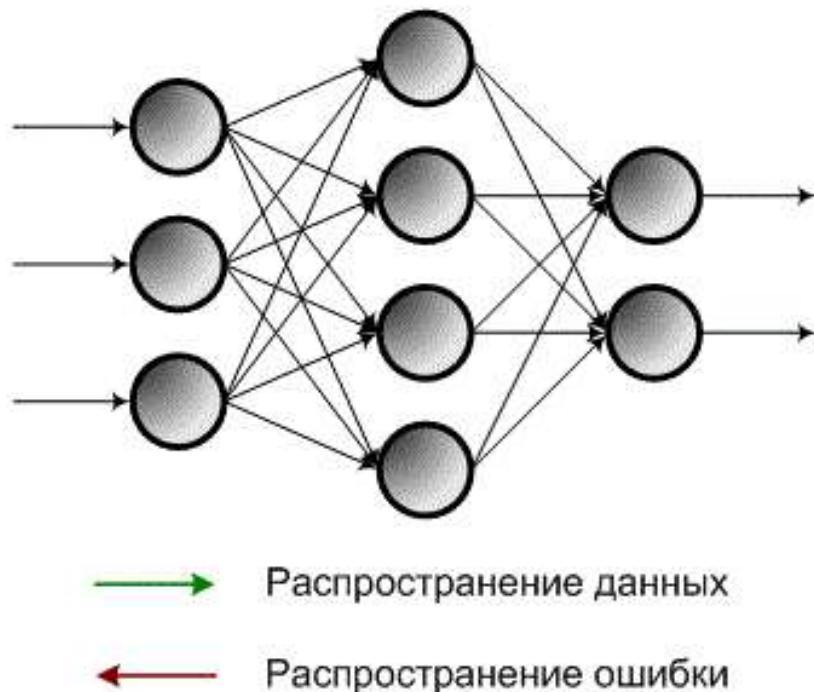
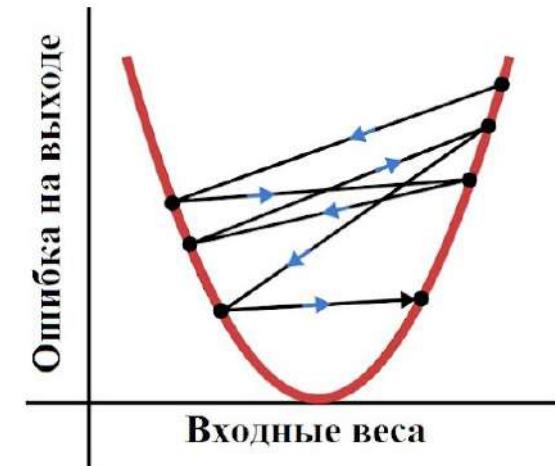
Градиентный спуск

- Мы не знаем на самом деле красивый график функции потерь
- Как будто спускаемся по оврагу с закрытыми глазами
- Поэтому считаем градиент в точке и обновляем веса

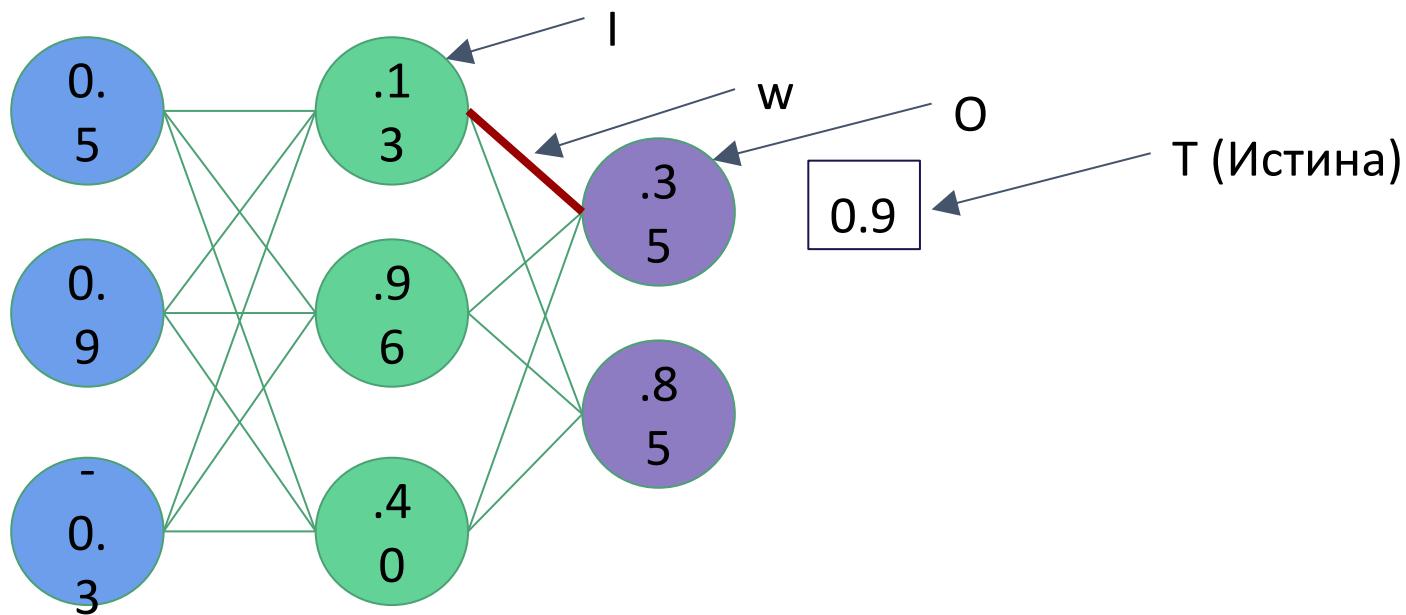


Обратное распространение ошибки

- Вопрос: какие веса должны быть обновлены и на сколько?
 - Подсказка: используйте производную от ошибки по отношению к весу, чтобы найти объем «вины»



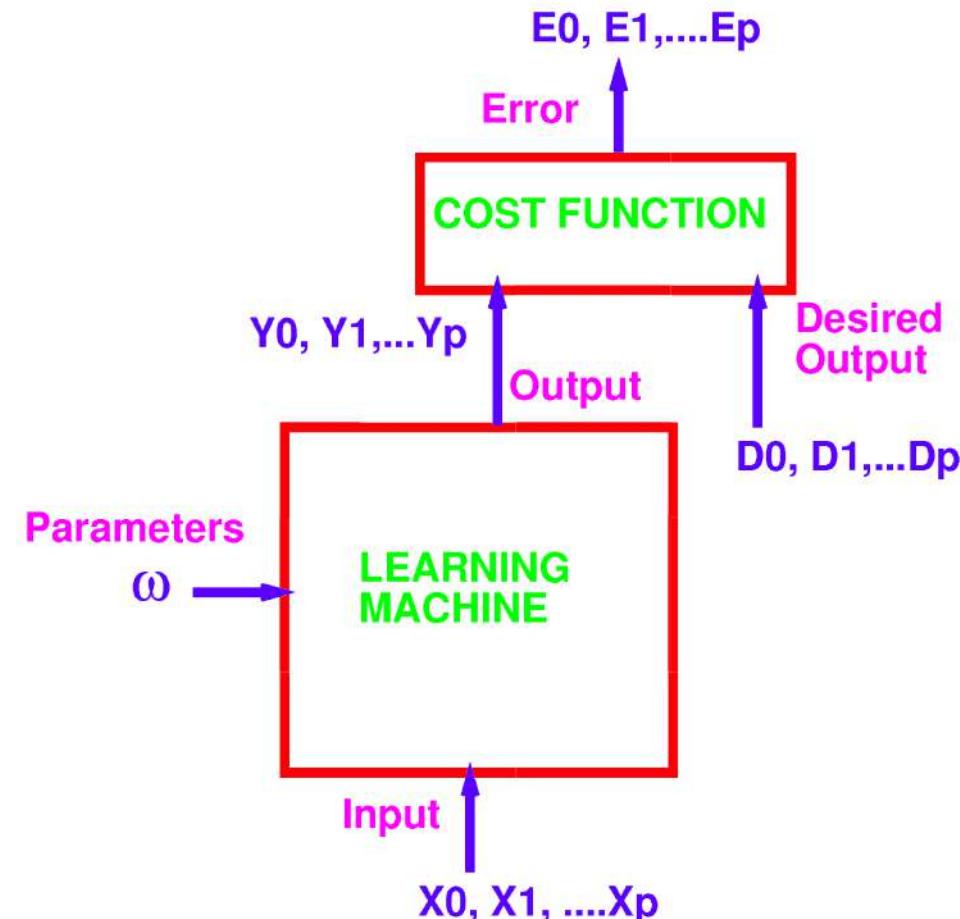
Пример обратного распространения



$$\frac{\partial E}{\partial w} = I \cdot (O - T) \cdot O \cdot (1 - O)$$

$$\frac{\partial E}{\partial w} = .13 \cdot (.35 - .9) \cdot .35 \cdot (1 - .35)$$

Основные понятия, терминология и обозначения

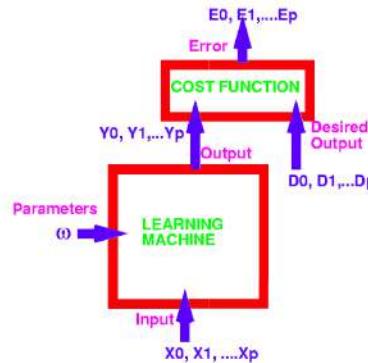


Average Error: $E = \frac{1}{p} \sum E_k$

Обучение градиентным спуском

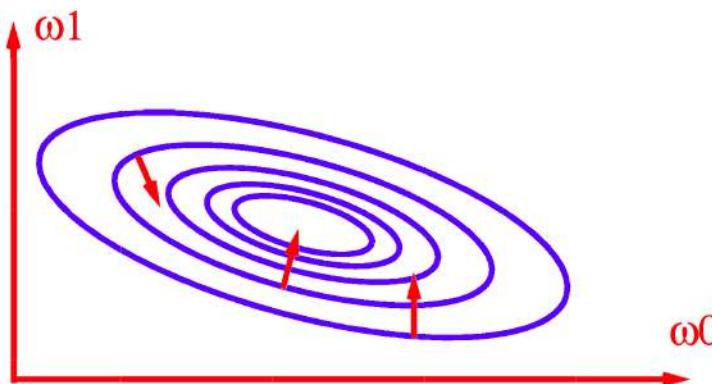
Average Error:

$$E(\omega) = \frac{1}{p} \sum E_k(\omega)$$

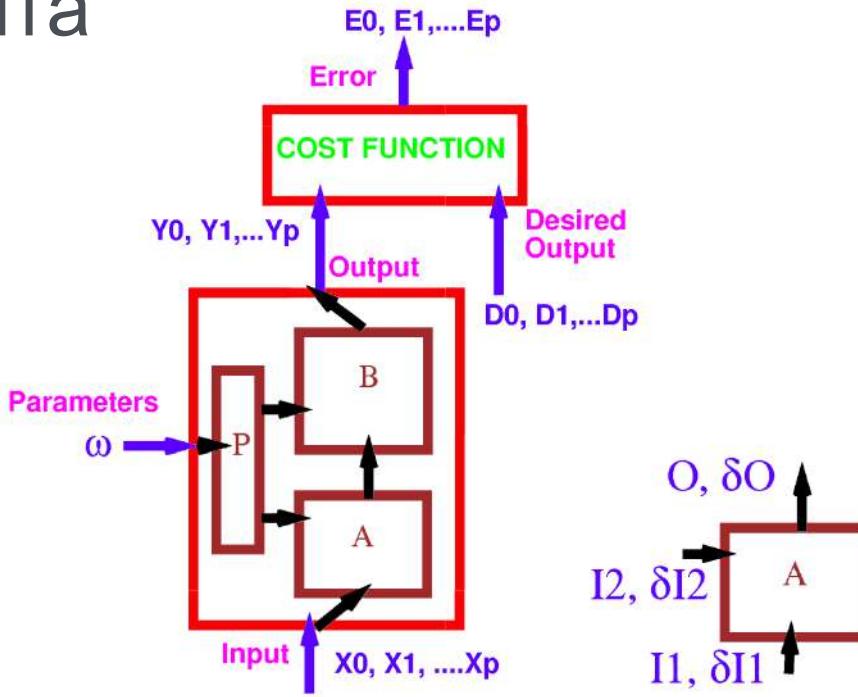


Gradient Descent:

$$\omega(\tau+1) = \omega(\tau) - \eta \frac{\partial E}{\partial \omega}$$



Вычисление градиента в обратном распространении



- Обучаемая модель состоит из модулей (например слои)
- Каждый модуль делает две процедуры:
 1. Вычисляет свои выходные значения из своих входных значений (пр. распр.)
- Каждый модуль делает две процедуры:
 1. Вычисляет градиентные векторы входных значений по градиентным векторам выходных значений (обр. распр.)

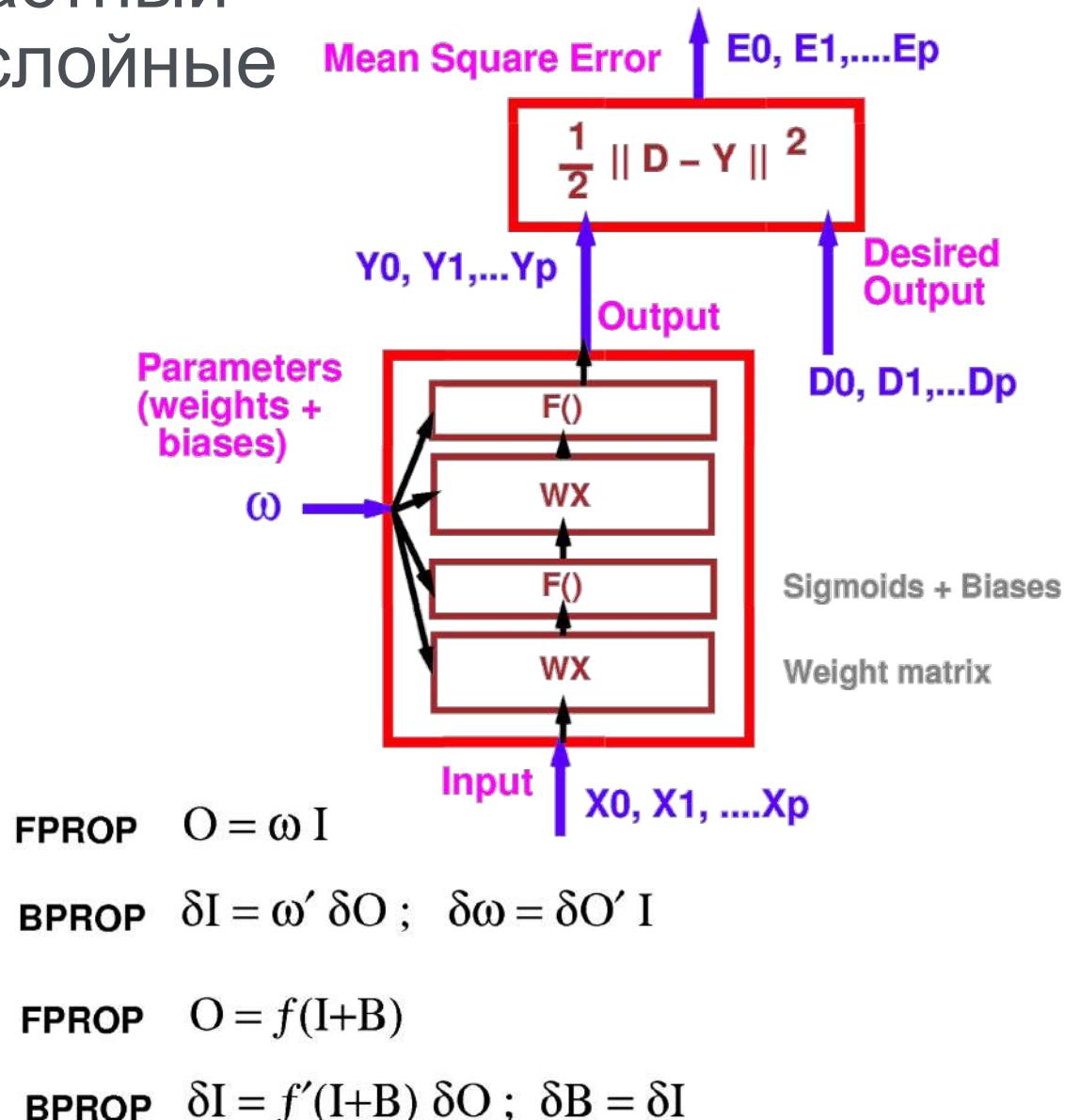
$$O = A(I_1, I_2)$$

2. Вычисляет градиентные векторы входных значений по градиентным векторам выходных значений (обр. распр.)

$$\delta I_1 = \frac{\partial A}{\partial I_1} \delta O$$

$$\delta I_2 = \frac{\partial A}{\partial I_2} \delta O$$

Интересный частный случай: многослойные сети



– Матрицы весов:

– Сигмоиды + отступ(bias):

Стохастичное обновление

- Стохастический градиент

$$\omega(\tau+1) = \omega(\tau) - \eta \frac{\partial E\tau}{\partial \omega}$$

```
Repeat {  
    for all examples in training set {  
        forward prop      // compute output  
        backward prop    // compute gradients  
        update parameters }}}
```

- Параметры обновляются после показа каждого примера

- Полный градиент

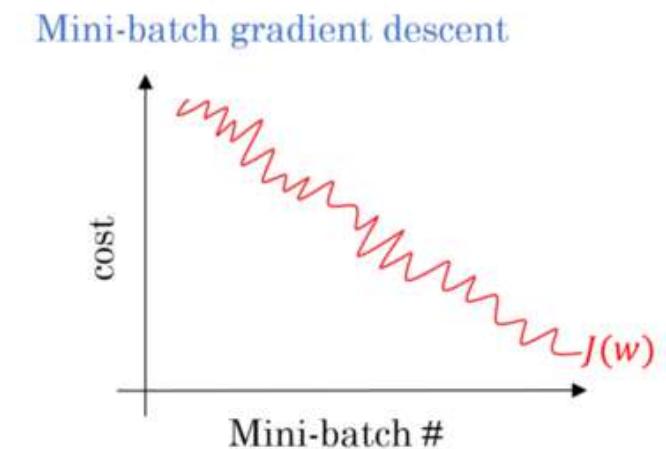
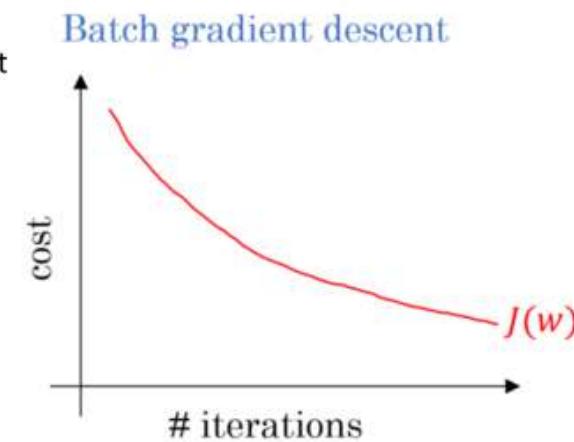
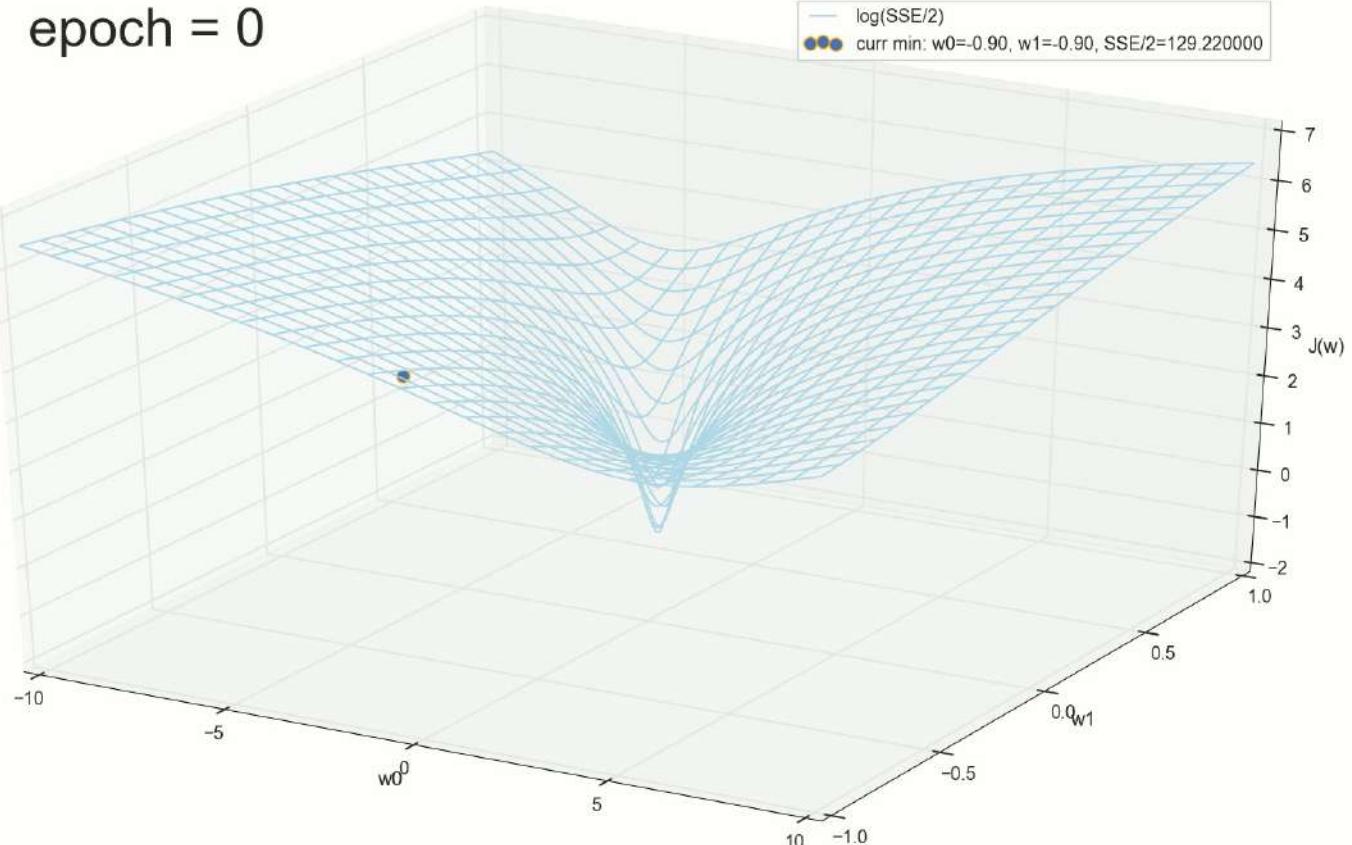
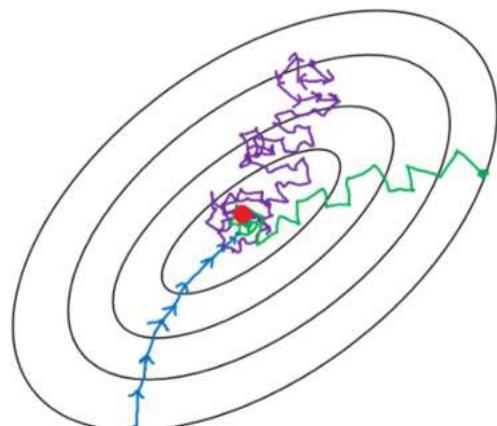
$$\omega(\rho+1) = \omega(\rho) - \eta \frac{\partial E}{\partial \omega}$$

```
Repeat {  
    for all examples in training set {  
        forward prop      // compute output  
        backward prop    // compute gradients  
        accumulate gradient }  
    update parameters }
```

- Перед обновлением параметра из всего обучающего набора накапливаются градиенты

БАТЧИ

- Функцию потерь считаем по одному примеру, но потом их складываем в одну
- Обучать на всем датасете – долго. Каждый раз берем небольшую порцию данных
- Но обучение становится хаотичнее



Итерации и эпохи

- Итерации – один шаг обучения
- Эпоха – обход всех экземпляров набора данных

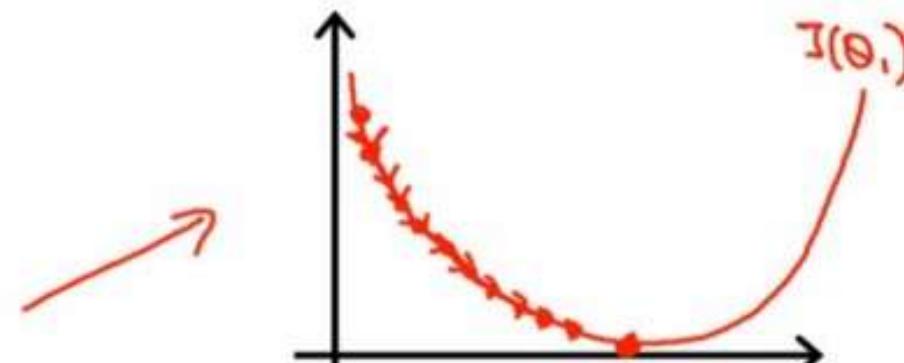


Скорость обучения

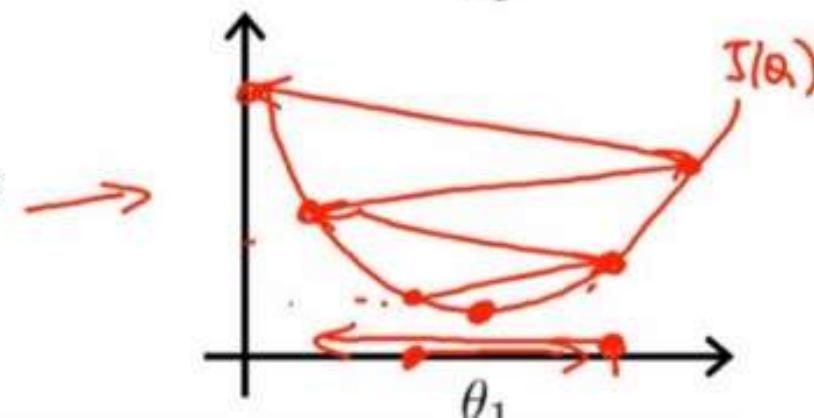
- Параметры (веса) сети меняются при обучении
- Гиперпараметры – нет. Управляем обучением

$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$

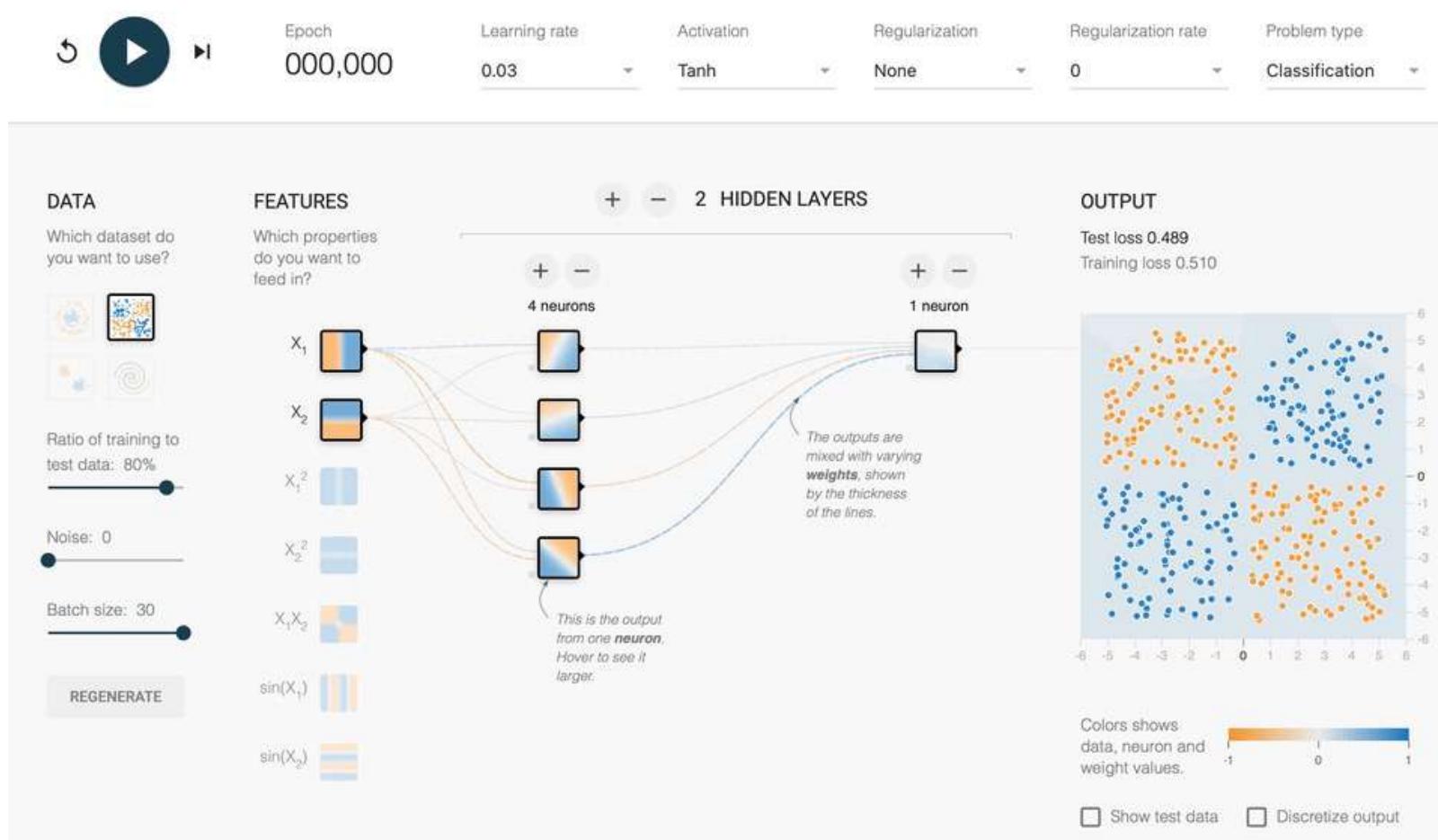
Если α слишком маленькая, то градиентный спуск может быть слишком медленным.



Если α слишком большая, то градиентный спуск может не попасть в точку минимума. Кроме того, сходимость может быть не достигнута.



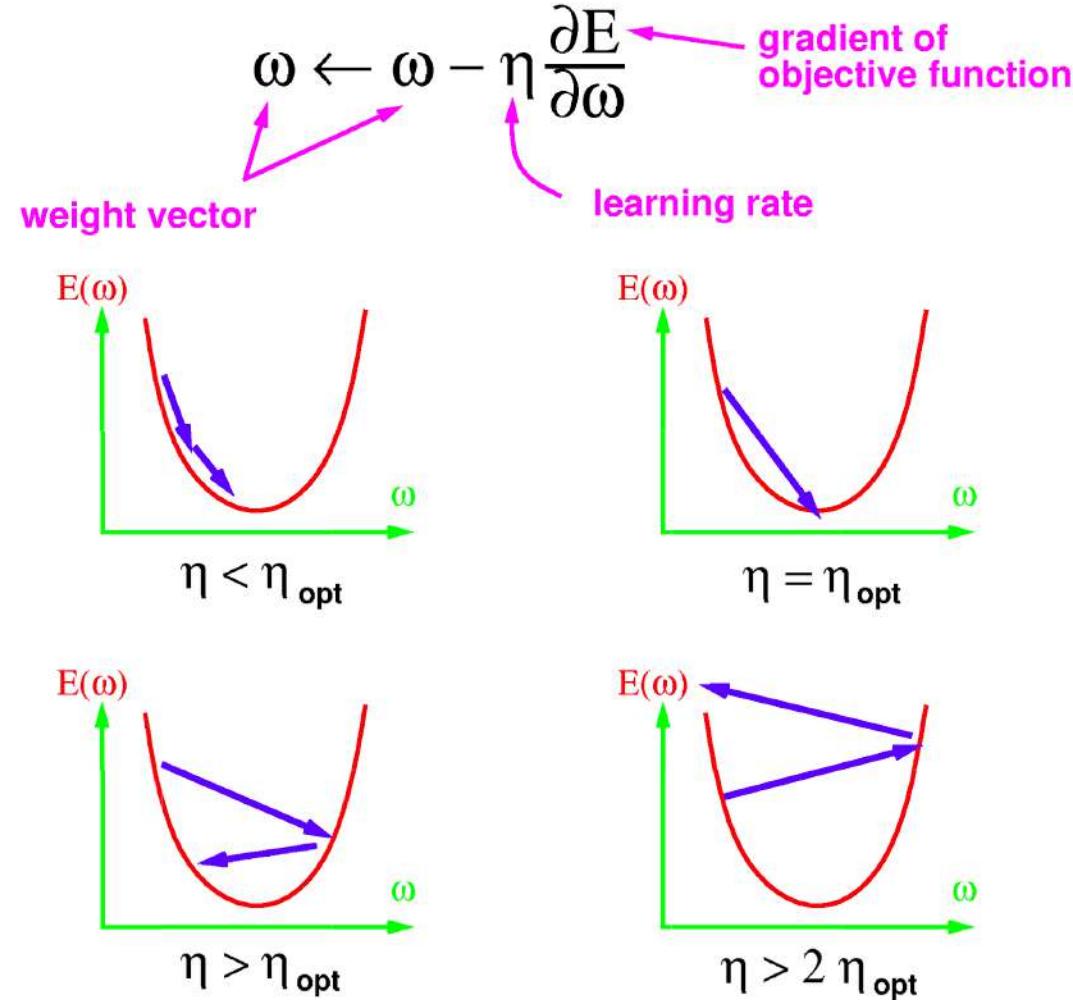
Визуализация обучения



<https://playground.tensorflow.org>

Немного теории

Градиентный спуск в одном измерении



Оптимальный шаг обучения в 1D

Weight change:

$$\Delta\omega = \eta \frac{\partial E}{\partial \omega}$$

Assuming E is quadratic:

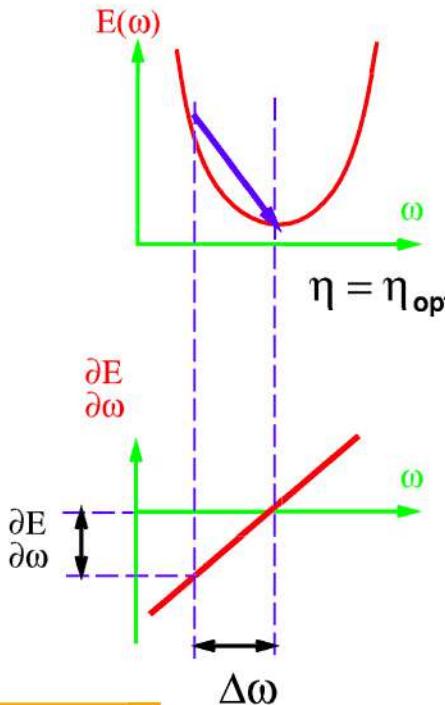
$$\frac{\partial^2 E}{\partial \omega^2} \Delta\omega = \frac{\partial E}{\partial \omega}$$

Optimal Learning Rate

$$\eta_{\text{opt}} = \left(\frac{\partial^2 E}{\partial \omega^2} \right)^{-1}$$

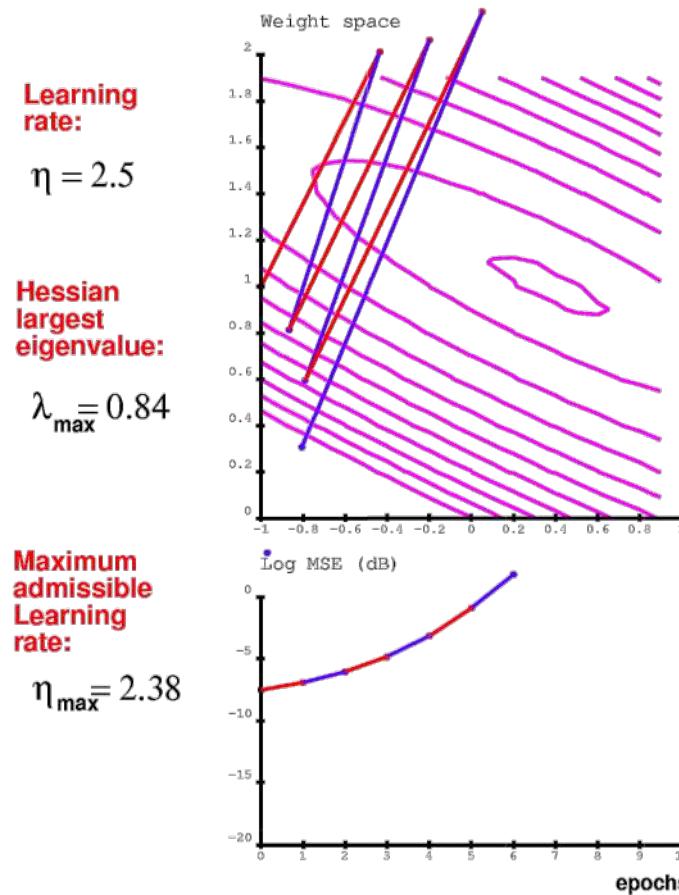
Maximum Learning Rate

$$\eta_{\text{max}} = 2 \eta_{\text{opt}}$$



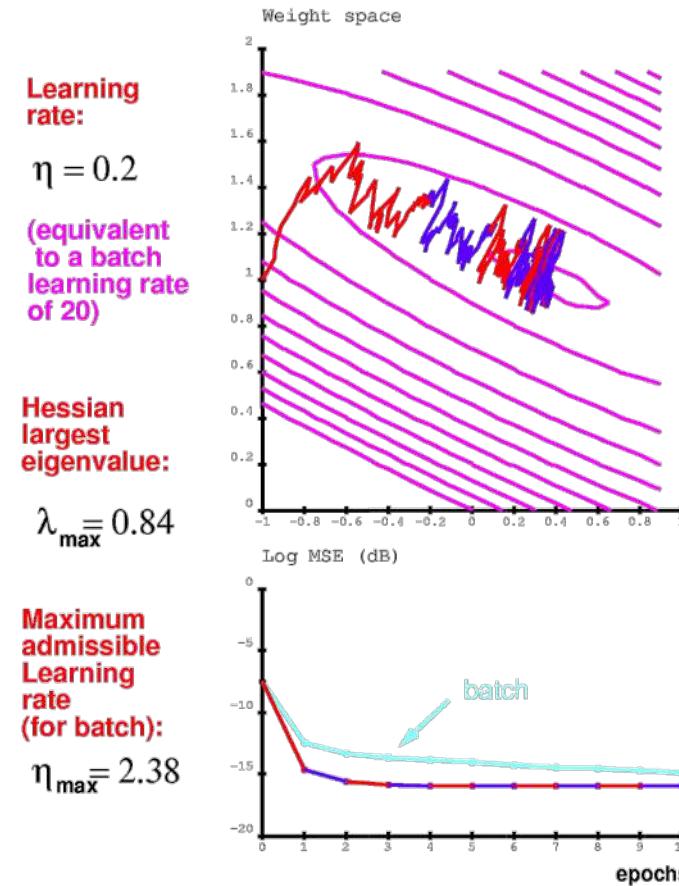
Пакетный градиентный спуск

- набор данных: набор-1 (100 примеров, 2 гауссова распределения) сеть: 1 линейный нейрон, 2 входа, 1 выход. 2 веса, 1 отступ.



Стохастический градиентный спуск

- набор данных: набор-1 (100 примеров, 2 гауссова распределения) сеть: 1 линейный нейрон, 2 входа, 1 выход. 2 веса, 1 отступ.



Стохастическое vs пакетное обновление

- Стохастическое обновление обычно НАМНОГО быстрее, чем пакетное обновление. Особенно на больших избыточных наборах данных.
- Вот почему:
 - Представьте что у вас есть обучающий набор из 1000 примеров.
 - Этот обучающий набор состоит из 10 экземпляров по 100 примеров.
- Пакет (батч): вычисление для одного обновления будет в 10 раз больше необходимого
- Стохастическое: будет использоваться избыточность в учебном наборе для получения преимущества обучения. Одна эпоха на большом наборе будет похожа на 10 эпох на меньшем наборе.
- Пакетное обновление будет КАК МИНИМУМ в 10 раз медленнее стохастичного
- В реальной жизни повторения редко происходят, но очень часто обучающие примеры очень избыточны (много примеров похожи друг на друга), что имеет тот же эффект.
- На практике нередки разницы скорости (на порядки) между пакетным и стохастическим обновлением.
- Маленькие пакеты могут использоваться без штрафа, если примеры в минибатче не слишком похожи.

Стохастическое vs пакетное обновление

Стохастическое

– Преимущества:

- Быстрая сходимость на больших избыточных данных
- Стохастическая траектория позволяет избежать локальных минимумов

– Недостатки:

- Продолжает «прыгать», если скорость обучения не уменьшается
- Теоретические условия сходимости не так понятны, как для пакетного обновления
- Доказательства сходимости вероятностны
- Большинство хороших способов ускорения или методов второго порядка не работают со стохастическим градиентом
- Сложнее распараллелить, чем пакетное обновление

Пакетное

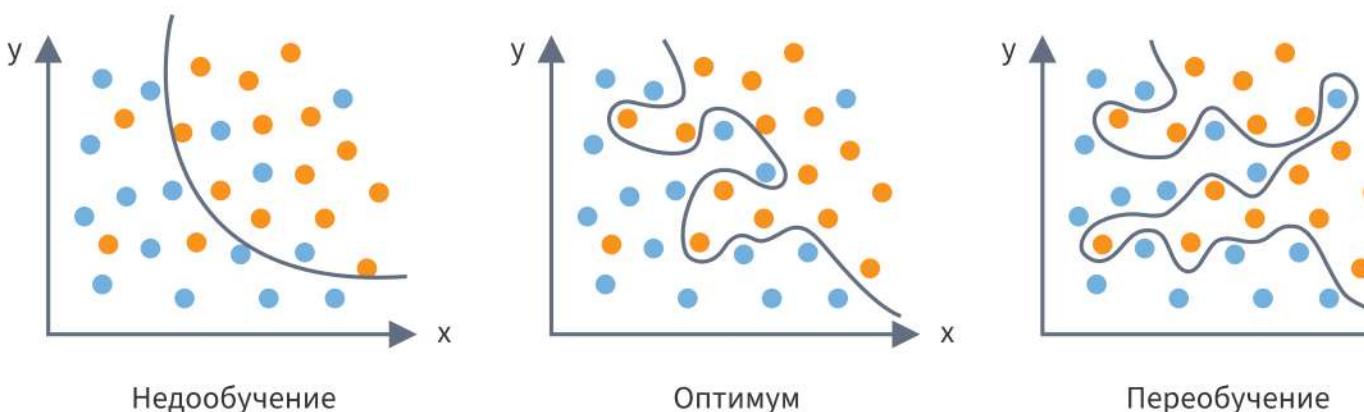
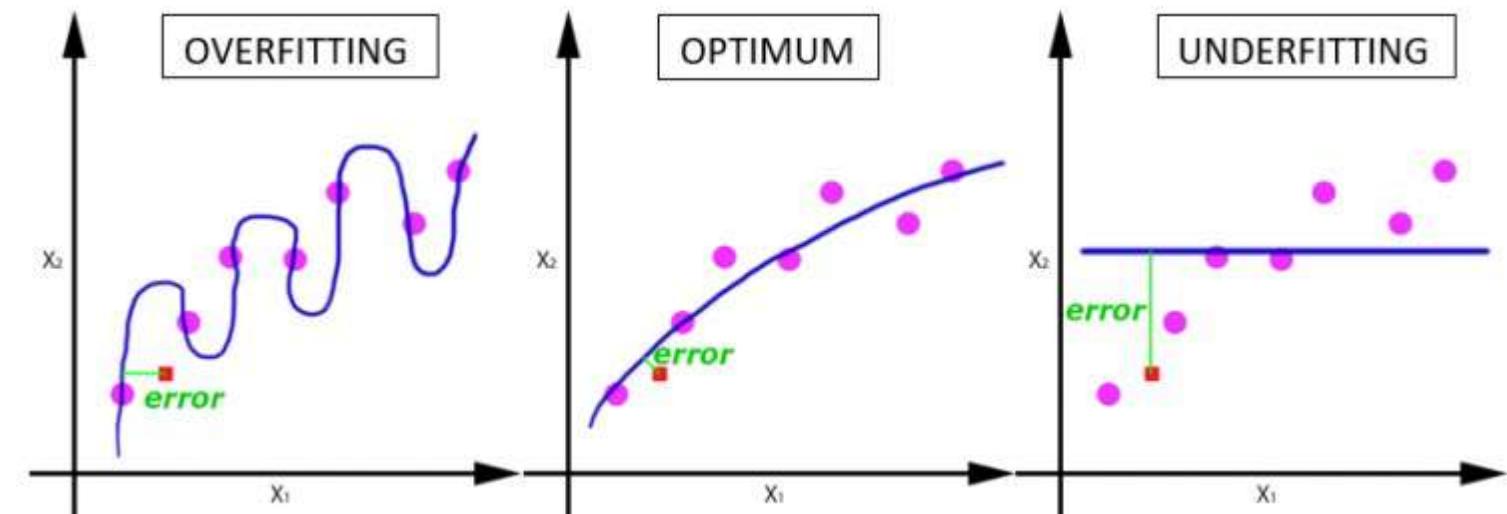
– Преимущества:

- Гарантированное сходимость к локальному минимуму в простых условиях
- Много способов и методов второго порядка для ускорения
- Простые доказательства сходимости

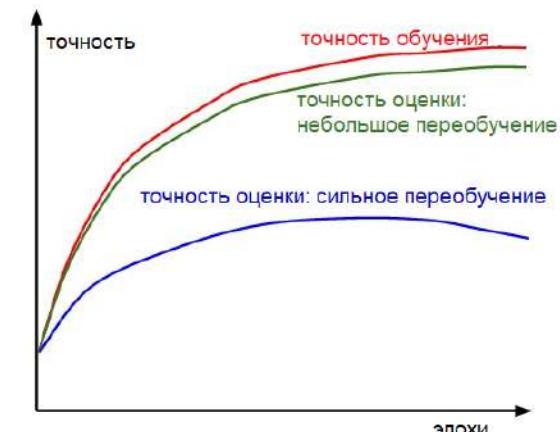
– Недостатки:

- Болезненно медленный на больших задачах
- Несмотря на длинный список недостатков для стохастического обновления, это то, что большинство людей используют (что справедливо, по крайней мере, для больших задач).

Точность, переобучение



- Переобучение при долгом обучении слишком сложной модели



Лекция 2

Сверточные нейронные сети

Разработка нейросетевых систем

Cifar100

- Набор данных, состоящий из цветных изображений 100 классов
- Размер 32 на 32 пикселя
- 3 цвета

airplane



automobile



bird



cat



deer



dog



frog



horse



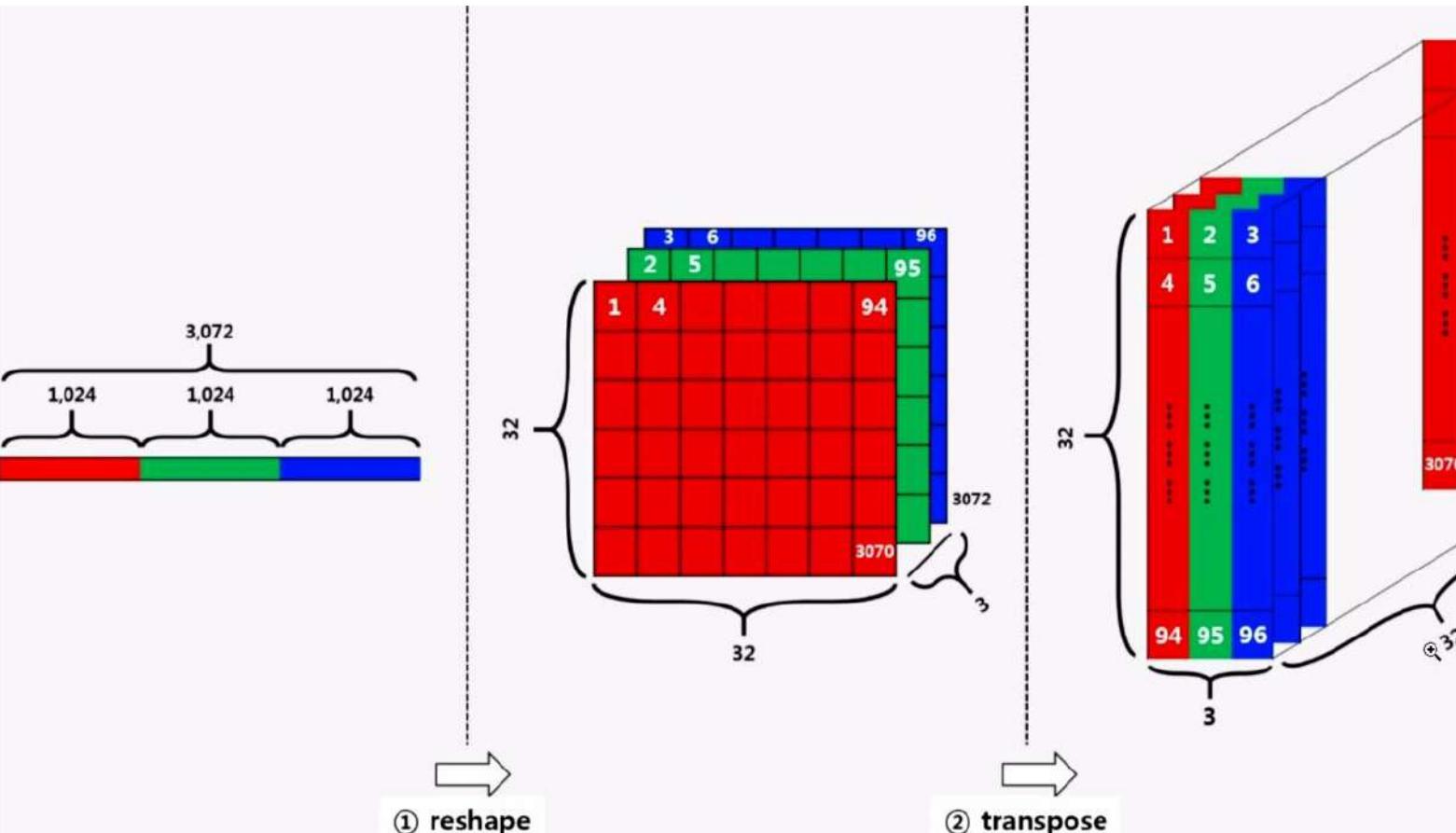
ship



truck



Reshape, transpose

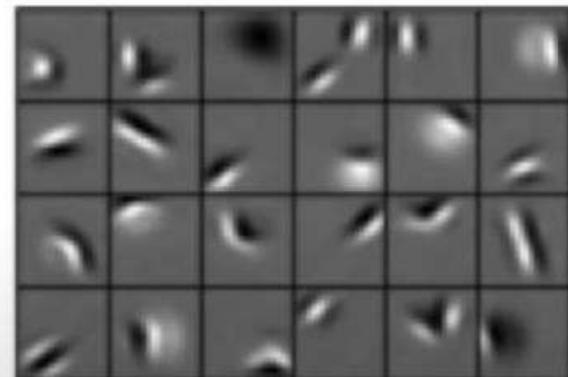


- Reshape – изменение размерности матрицы
- Transpose – транспонирование
- Количество элементов в матрице остается прежним - 3072

Из биологии

- Идея сверточных нейронных сетей навеяна нам из реального мира
- Зрение человека ищет нужны набор признаков вокруг, воспринимает их и из простых признаков собирает сложные

Low level features



Edges, dark spots

Mid level features



Eyes, ears, nose

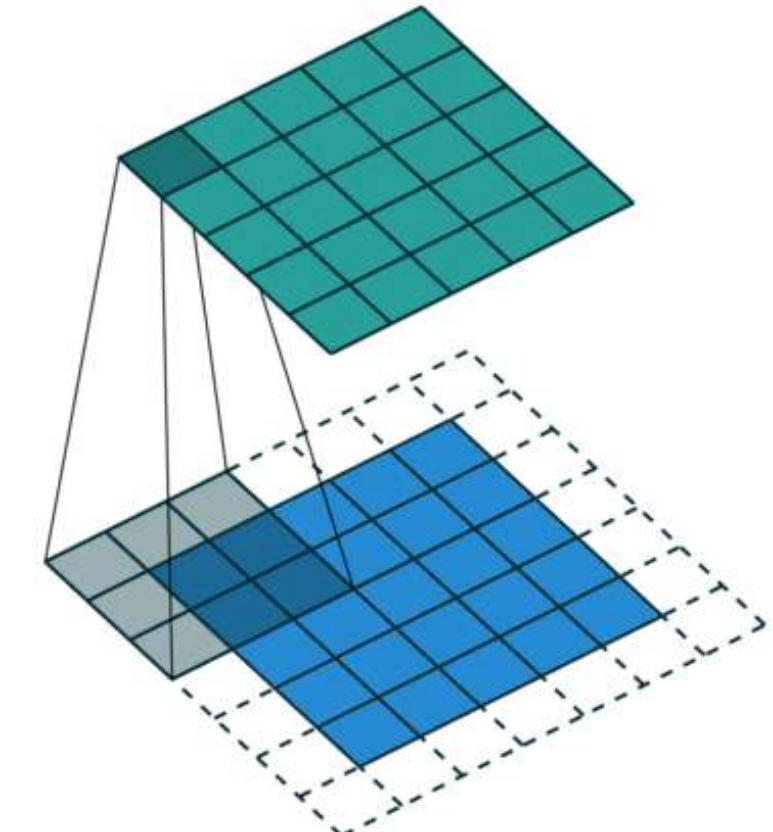
High level features



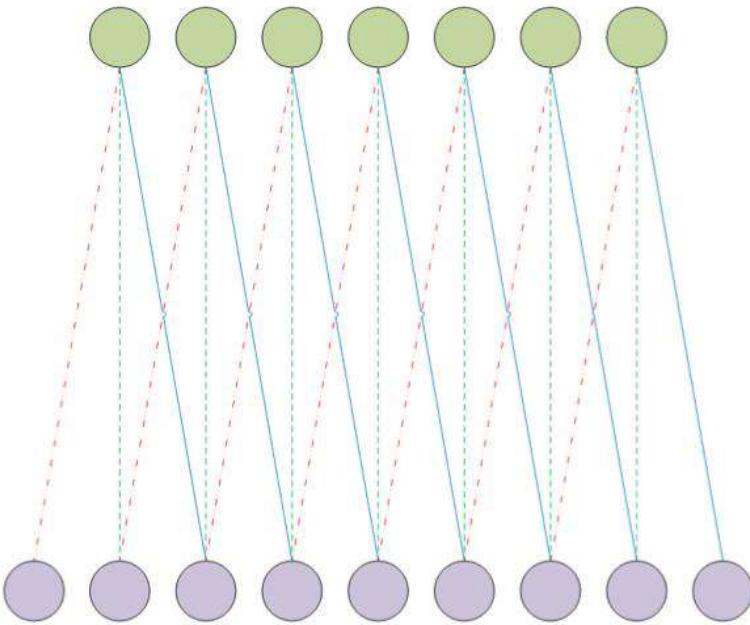
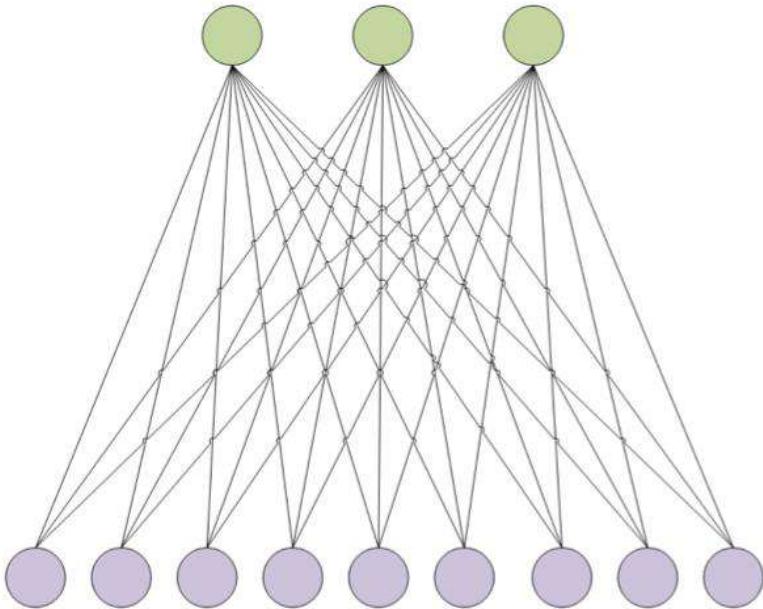
Facial structure

Свертка

- Свертка – это применение одного и того же фильтра (нейрона) к разным частям исходного изображения
- В результате на разных частях исходных данных идет поиск одинаковых признаков
- Выходными данными свертки являются карты признаков
- Несколько фильтров позволяют сформировать несколько карт признаков одного сверточного слоя



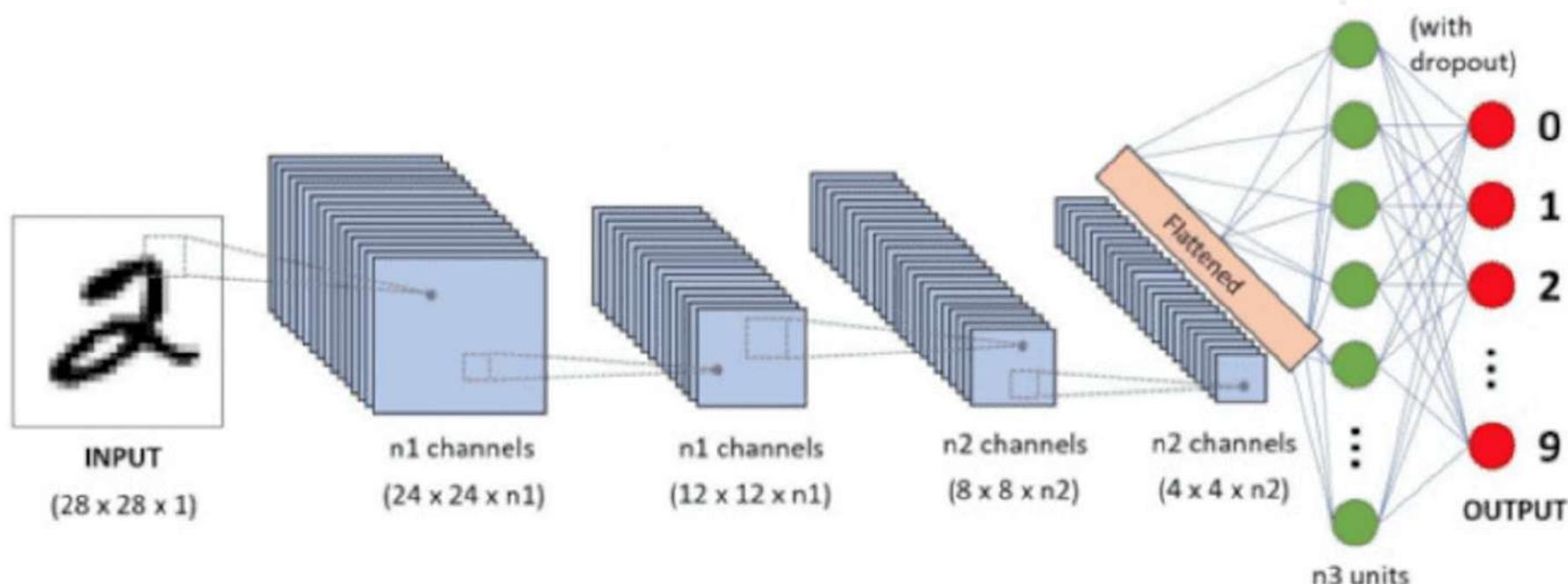
Свойства сверточного слоя



- Разреженные взаимодействия – каждый нейрон связан с ограниченным числом входных нейронов
- Разделение параметров – в карте признаков все нейроны имеют одинаковый набор параметров
- Инвариантность – сдвиг исходных данных вызывает аналогичный сдвиг значений выходного слоя

Сверточная нейросеть

- Сверточная нейросеть состоит из нескольких слоев: свертки, пуллинга, полно связного
- Слои свертки и пуллинга чередуются друг за другом
- Слои свертки применяют набор n_1, n_2 фильтров к исходному изображению. Каждый фильтр ищет определенные признаки в исходных данных и формируется карта признаков
- Слои свертки обучаются, меняют количество каналов. Вход для свертки трехмерный: ядро * ядро * каналы
- Слои пуллинга только уменьшают размерность карты признаков, количество каналов сохраняется.
- Данные последнего слоя пуллинга преобразуются в вектор для использования в полно связном слое



Вычисления в сверточном слое

0	0	0	0	0	0	0	...
0	156	155	156	158	158	158	...
0	153	154	157	159	159	159	...
0	149	151	155	158	159	159	...
0	146	146	149	153	158	158	...
0	145	143	143	148	158	158	...
...

Input Channel #1 (Red)

0	0	0	0	0	0	0	...
0	167	166	167	169	169	169	...
0	164	165	168	170	170	170	...
0	160	162	166	169	170	170	...
0	156	156	159	163	168	168	...
0	155	153	153	158	168	168	...
...

Input Channel #2 (Green)

0	0	0	0	0	0	0	...
0	163	162	163	165	165	165	...
0	160	161	164	166	166	166	...
0	156	158	162	165	166	166	...
0	155	155	158	162	167	167	...
0	154	152	152	157	167	167	...
...

Input Channel #3 (Blue)

- Трехмерный случай для трех цветов

-1	-1	1
0	1	-1
0	1	1

Kernel Channel #1

1	0	0
1	-1	-1
1	0	-1

Kernel Channel #2

0	1	1
0	1	0
1	-1	1

Kernel Channel #3



308

+



-498

+

164

+ 1 = -25

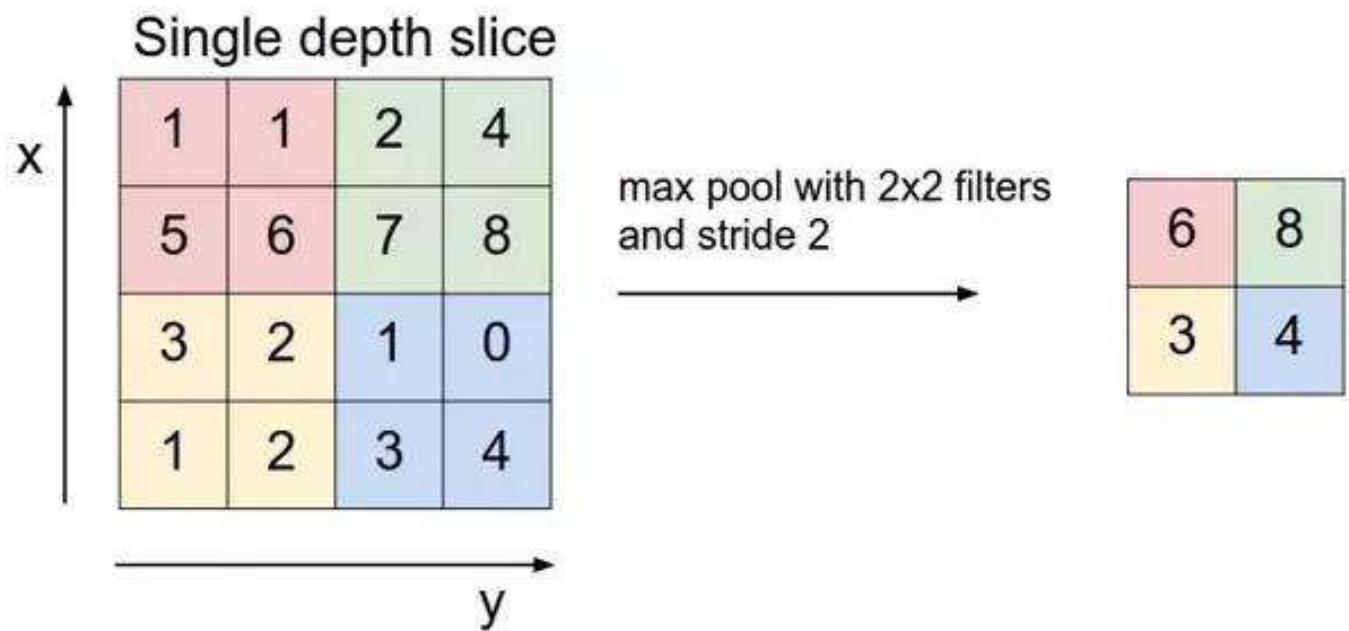
Bias = 1

-25				...
				...
				...
				...
...

- Аналогично несколько каналов слоя свертки
- На выходе нейрона 27 значений

Пуллинг

- Слой пуллинга позволяет сократить размерность карты признаков
- Первый вариант пуллинга - максимальное значение из нескольких соседних



Average Pooling

- Второй вариант вычисления – среднее значение из нескольких соседних

2	2	7	3
9	4	6	1
8	5	2	4
3	1	2	6

Average Pool
→
Filter - (2 x 2)
Stride - (2, 2)

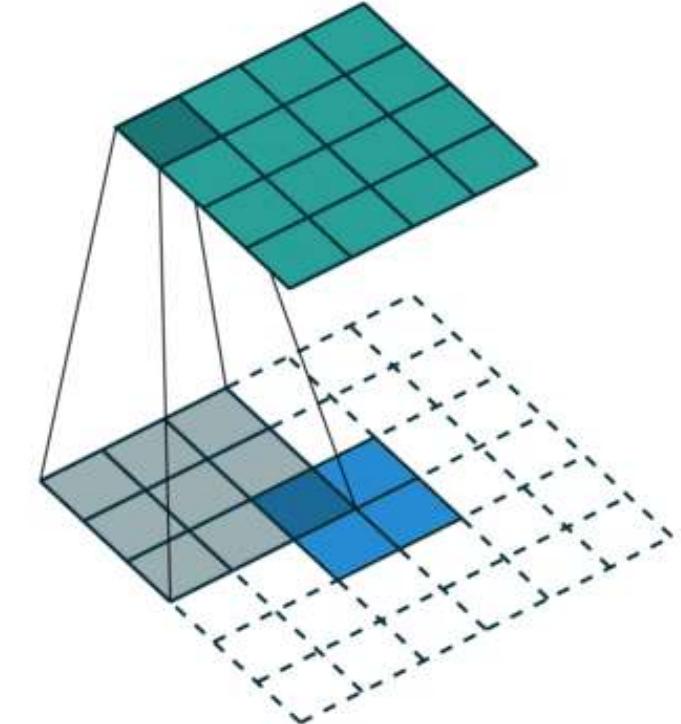
4.25	4.25
4.25	3.5

Padding

- Padding – заполнение исходных данных для свертки
- Либо нулями, либо повторение соседних ячеек
- Padding – возможность сохранить размерность карты признаков или даже ее увеличить
- Для сохранения размера, padding равен **(размеру ядра свертки-1)/2**

0 ₂	0 ₀	0 ₁	0	0	0	0
0 ₁	2 ₀	2 ₀	3	3	3	0
0 ₀	0 ₁	1 ₁	3	0	3	0
0	2	3	0	1	3	0
0	3	3	2	1	2	0
0	3	3	0	2	3	0
0	0	0	0	0	0	0

1	6	5
7	10	9
7	10	8

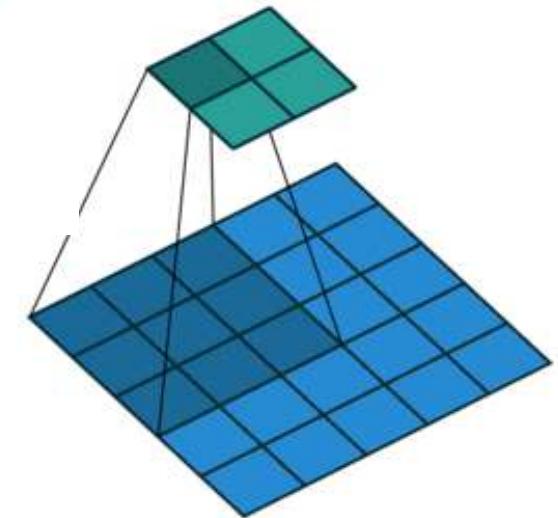


Шаг свертки - stride

- Stride - шаг свертки
 - Это регулируемый параметр, который определяет размерность карты признаков

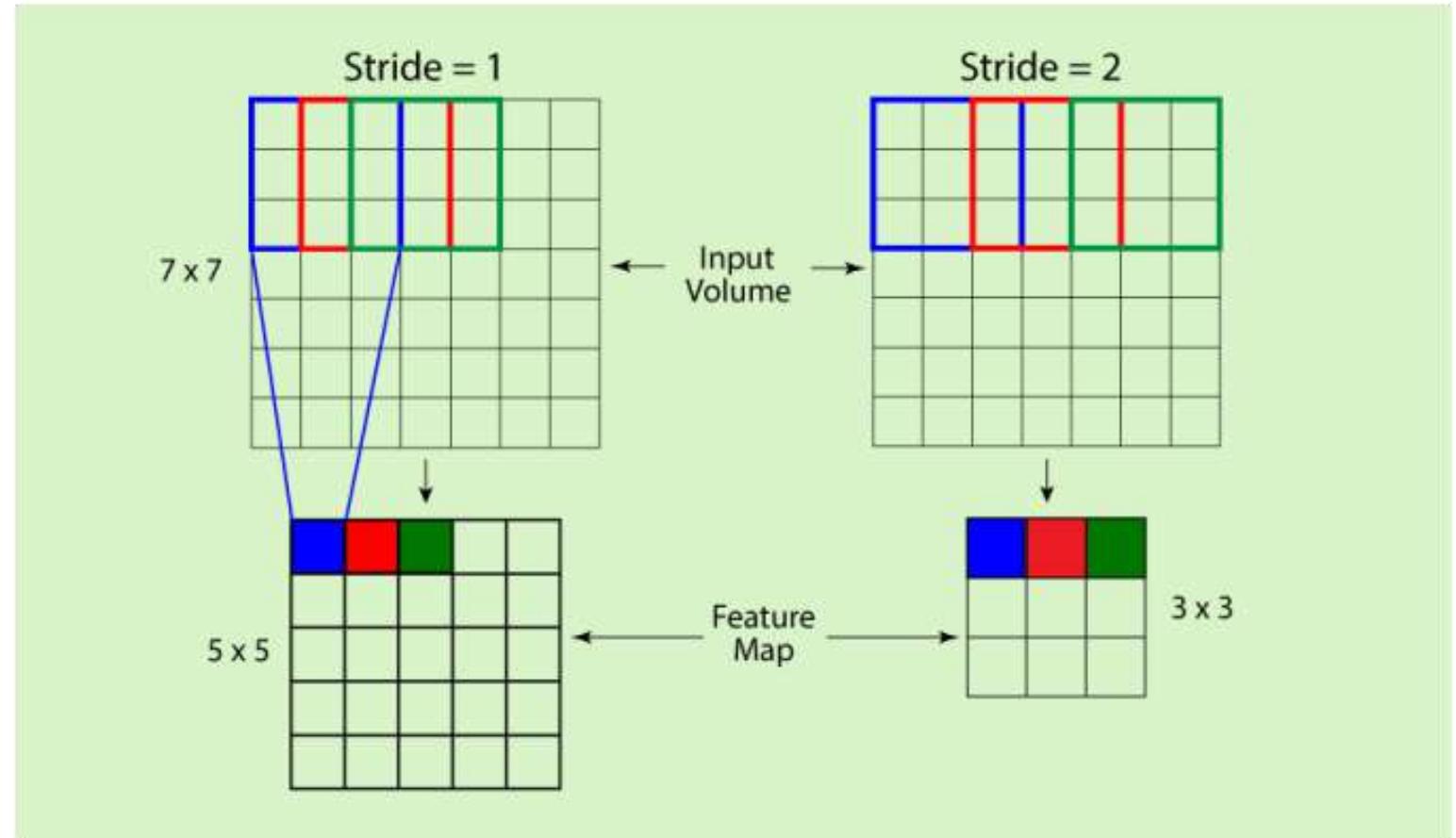
0 ₂	0 ₀	0 ₁	0	0	0	0
0 ₁	2 ₀	2 ₀	3	3	3	0
0 ₀	0 ₁	1 ₁	3	0	3	0
0	2	3	0	1	3	0
0	3	3	2	1	2	0
0	3	3	0	2	3	0
0	0	0	0	0	0	0

1	6	5
7	10	9
7	10	8

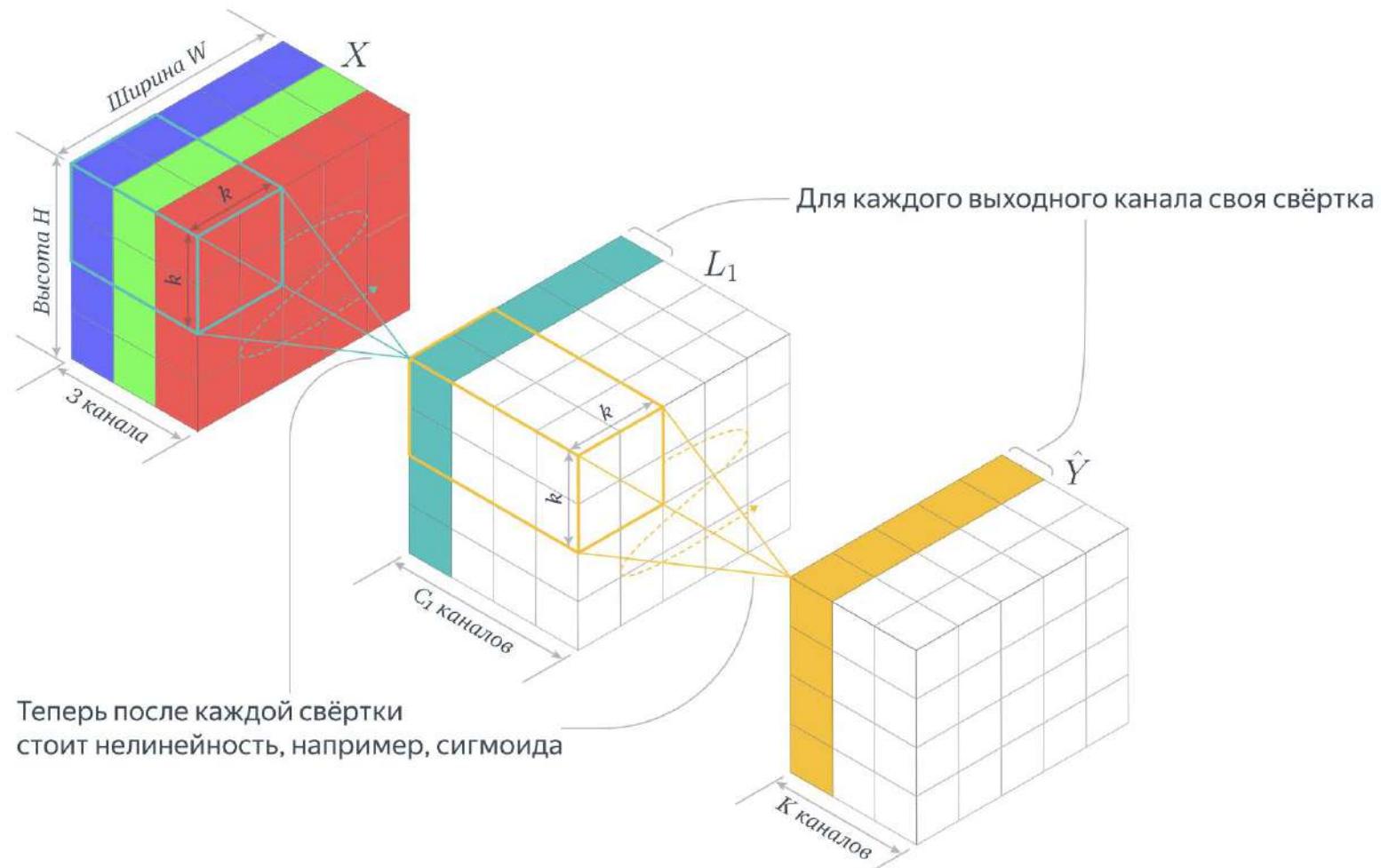


Stride

- Stride - это еще один способ сократить размерность карты признаков
- Чем больше шаг, тем меньше становится итоговая карта признаков

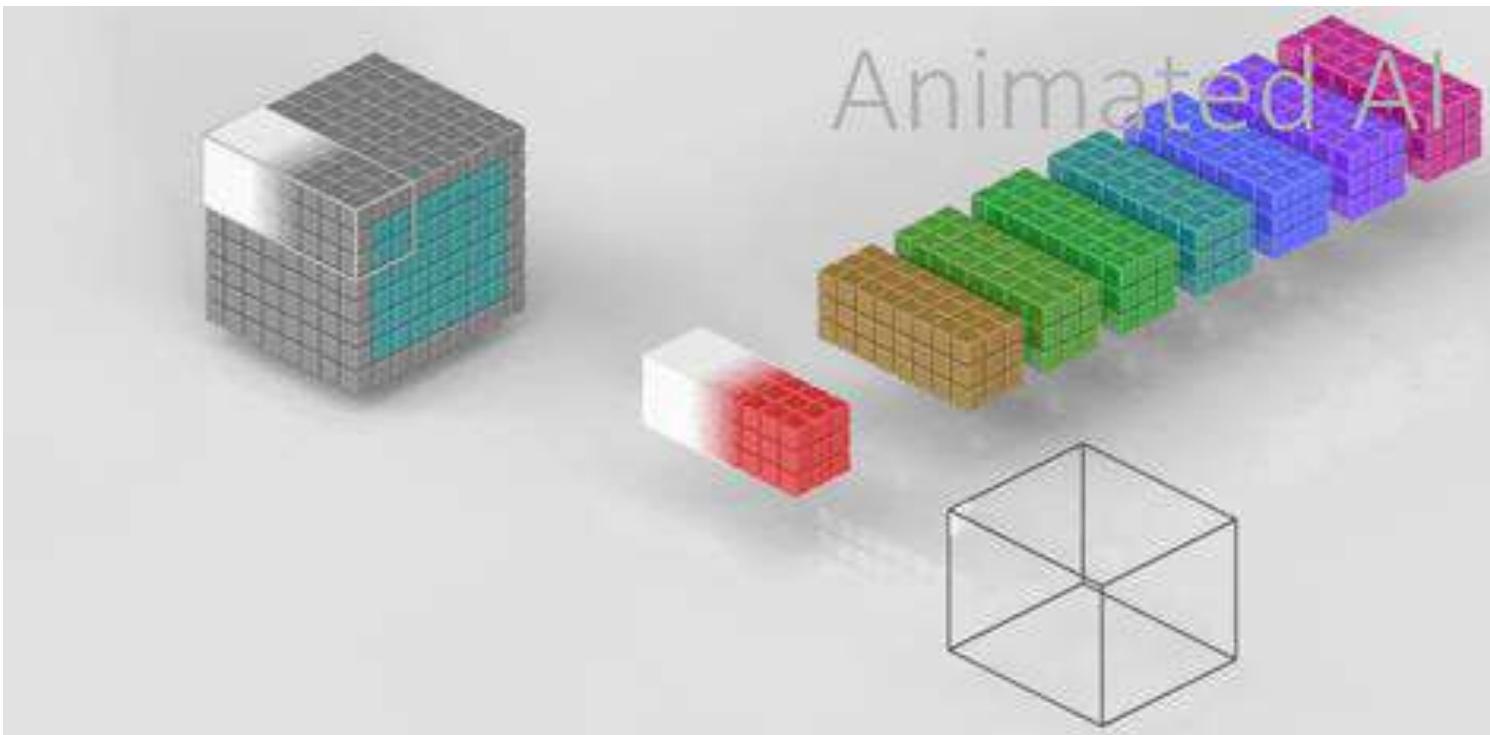


Хорошая статья по CNN



<https://education.yandex.ru/handbook/ml/article/svyortochnye-nejroseti>

Визуализация CNN



<https://animatedai.github.io>

Пример

```
def __init__(self, hidden_size=32, classes=100):
    super(Cifar100_MLP, self).__init__()
    # https://blog.jovian.ai/image-classification-of-cifar100-dataset-using-pytorch-8b7145242df1
    self.seq = nn.Sequential(
        Normalize([0.5074, 0.4867, 0.4411], [0.2011, 0.1987, 0.2025]),
        # первый способ уменьшения размерности картинки - через stride
        nn.Conv2d(3, HIDDEN_SIZE, 5, stride=4, padding=2),
        nn.ReLU(),
        # второй способ уменьшения размерности картинки - через слой пуллинг
        nn.Conv2d(HIDDEN_SIZE, HIDDEN_SIZE*2, 3, stride=1, padding=1),
        nn.ReLU(),
        nn.AvgPool2d(4),#nn.MaxPool2d(4),
        nn.Flatten(),
        nn.Linear(HIDDEN_SIZE*8, classes),
    )
```

Layer (type)	Output Shape	Param #
Normalize-1	[-1, 3, 32, 32]	0
Conv2d-2	[-1, 32, 8, 8]	2,432
ReLU-3	[-1, 32, 8, 8]	0
Conv2d-4	[-1, 64, 8, 8]	18,496
ReLU-5	[-1, 64, 8, 8]	0
AvgPool2d-6	[-1, 64, 2, 2]	0
Flatten-7	[-1, 256]	0
Linear-8	[-1, 3]	771

Результаты обучения

- Точность классификации удалось значительно повысить
- Однако точность на тестовой выборке пока значительно уступает точности на обучающей
- Нейросеть “запомнила” примеры с обучающей выборки

	train				
	precision	recall	f1-score	support	
0	1.0000	1.0000	1.0000	500	
55	1.0000	1.0000	1.0000	500	
58	1.0000	1.0000	1.0000	500	
accuracy			1.0000	1500	
macro avg	1.0000	1.0000	1.0000	1500	
weighted avg	1.0000	1.0000	1.0000	1500	

	test				
	precision	recall	f1-score	support	
0	0.9238	0.9700	0.9463	100	
55	0.8515	0.8600	0.8557	100	
58	0.9043	0.8500	0.8763	100	
accuracy			0.8933	300	
macro avg	0.8932	0.8933	0.8928	300	
weighted avg	0.8932	0.8933	0.8928	300	

Стохастический градиентный спуск

$L(f(\mathbf{x}(i); \boldsymbol{\theta}), y(i))$ – значение функции потерь

$f(\mathbf{x}(i); \boldsymbol{\theta})$ – результат вычисления нейронной сети от входа $\mathbf{x}(i)$ и параметров (весов) $\boldsymbol{\theta}$

Обновление на k -ой итерации стохастического градиентного спуска (СГС)

Require: скорость обучения ϵ_k

Require: Начальные значения параметров $\boldsymbol{\theta}$

while критерий остановки не выполнен **do**

Выбрать из обучающего набора мини-батч m примеров $\{\mathbf{x}(1), \dots, \mathbf{x}(m)\}$ и соответствующие им метки $y(i)$.

Вычислить оценку градиента: $g \leftarrow + (1/m) \nabla_{\boldsymbol{\theta}} \sum_i L(f(\mathbf{x}(i); \boldsymbol{\theta}), y(i))$.

Применить обновление: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon g$.

end while

Стохастический градиентный спуск

- Основной параметр алгоритма СГС – скорость обучения ϵ
- На практике же необходимо постепенно уменьшать скорость обучения со временем
- $\epsilon_k = (1 - \alpha) \epsilon_{k-1} + \alpha \epsilon_k$, где $\alpha = k / \tau$. После τ -й итерации ϵ остается постоянным.
- Если скорость изменяется линейно, то нужно задать параметры $\epsilon_0, \epsilon_\tau$ и τ .

Импульсный метод

Стохастический градиентный спуск (СГС) с учетом импульса

Require: скорость обучения ε , параметр импульса α

Require: начальные значения параметров θ , начальная скорость v

while критерий остановки не выполнен **do**

Выбрать из обучающего набора мини-пакет m примеров $\{x(1), \dots, x(m)\}$ и соответствующие им метки $y(i)$.

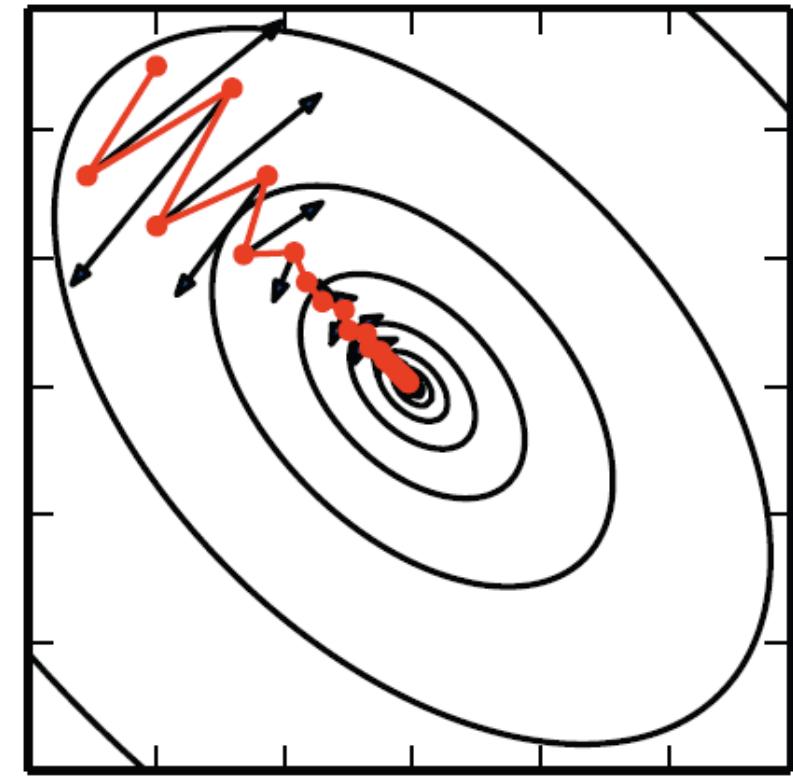
Вычислить оценку градиента:

$$g \leftarrow (1/m) \nabla_{\theta} \sum_i L(f(x(i); \theta), y(i)).$$

Вычислить обновление скорости: $v \leftarrow \alpha v - \varepsilon g$.

Применить обновление: $\theta \leftarrow \theta + v$.

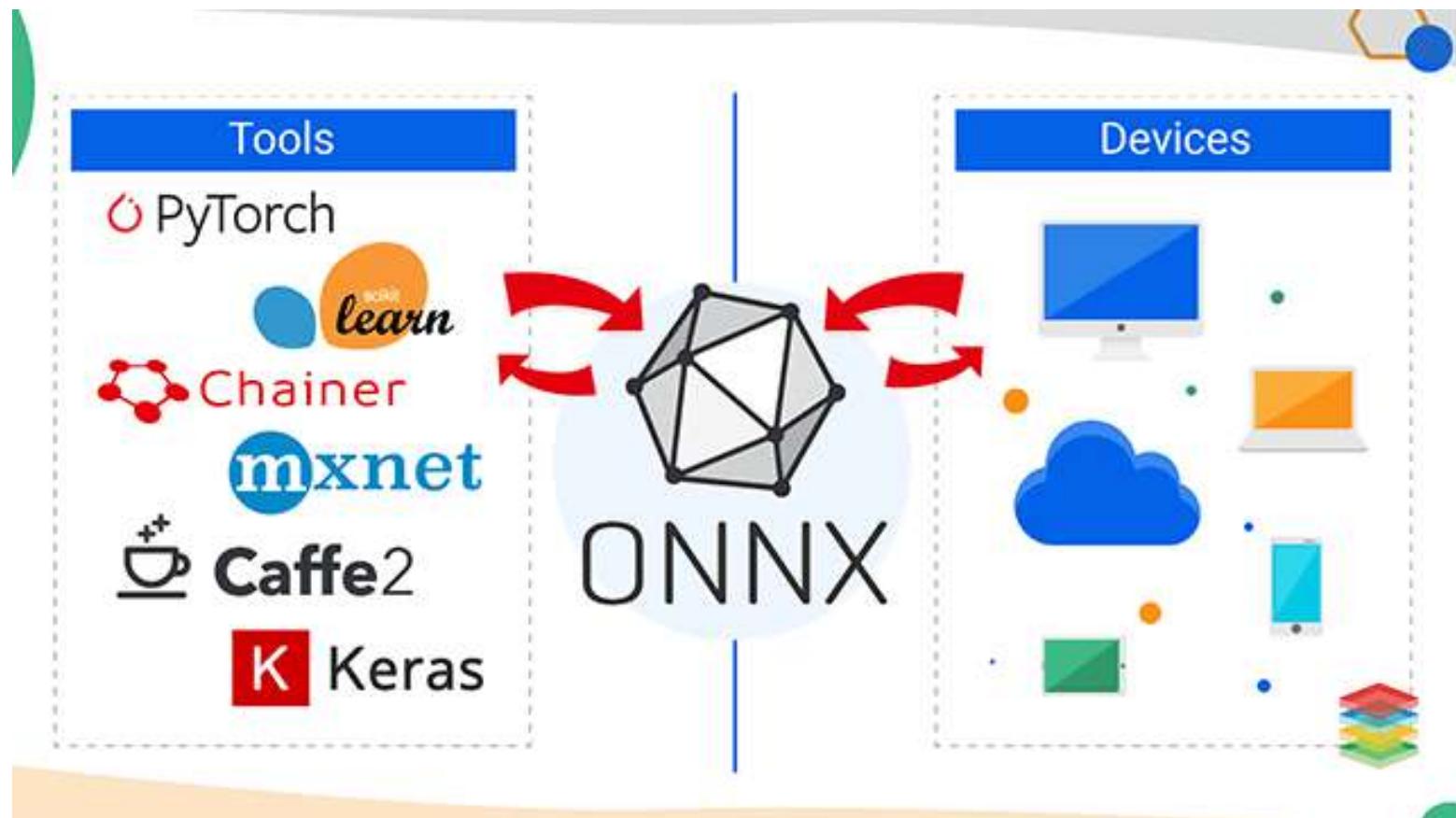
end while



Импульсный алгоритм можно рассматривать как имитацию движения частицы, подчиняющейся динамике Ньютона.

ONNX

- ONNX - библиотека для конвертации моделей между разными технологиями
- ONNX дает возможность исследователям и разработчикам выбрать нужную комбинацию инструментов для решения задачи
- ONNX.js позволяет запускать модели в браузере, то есть на стороне пользователя



ONNX

Step 3. Select class labels and get predictions

Выбрать файл cifar100_CNN.onnx

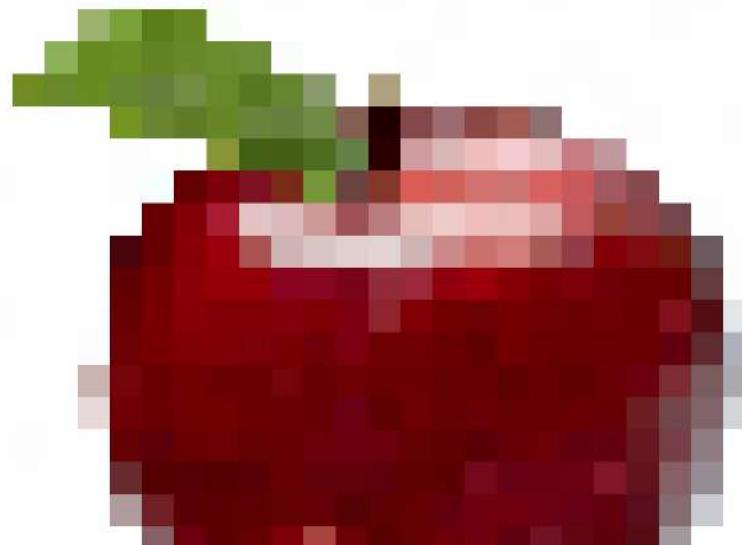
Select ONNX file

Выбрать файл c25c94fe96_1000.jpg

Class label 54 ⚠ Add 10 Random Undo Reset

0,50,54

← ↑ → ↓



Лекция 3

Регуляризация и аугментация данных

Разработка нейросетевых систем

Итог по сверточным нейронным сетям

Анализ

- CNN повышает точность на тестовой выборке
- Но точность на обучающей выборке по-прежнему выше точности на тестовой

Решение

- Разнообразие набора данных помогает модели обобщать



Аугментация данных



Аугментация данных

Различают две стратегии аугментации: offline-аугментация и online-аугментация. Первый вид предполагает заблаговременное расширение обучающей выборки за счёт добавления в неё преобразованных и искаженных копий.

Так, например, если отразить изображения по горизонтали и вертикали, получится увеличить объём обучающей выборки в 4 раза:

- 1x оригинальный датасет
- 2x - оригинальный + горизонтальное отражение
- 3x - оригинальный + горизонтальное отражение + вертикальное отражение
- 4x - оригинальный + горизонтальное отражение + вертикальное отражение + оба отражения одновременно

После расширения обучающей выборки она используется для обучения.

Offline-аугментация

Такой подход имеет следующие недостатки:

- Применяется лишь ограниченное число преобразований из числа возможных (например, при использовании преобразования поворота на целое число градусов)
- Увеличивается размер выборки, который необходимо держать в оперативной памяти или на жестком диске. Для больших наборов данных с миллионами примеров это непозволительная роскошь.

Есть и преимущество:

- Преобразования выполняются лишь раз, поэтому при обучении нейросети доступно больше вычислительных ресурсов.

Online-аугментация

Такой подход наоборот имеет следующие преимущества:

- Применяется каждую эпоху и итерацию новое случайное преобразование с заданными ограничениями (например, при использовании преобразования поворота на целое число градусов)
- Не нужно тратить память на дублирование данных – храним только оригиналы.

Недостаток:

- Преобразования выполняются каждый раз, отнимаем ресурсы от обучения. Но мы можем использовать для обучения GPU, а для аугментации – CPU.

Flipping- переворачивание

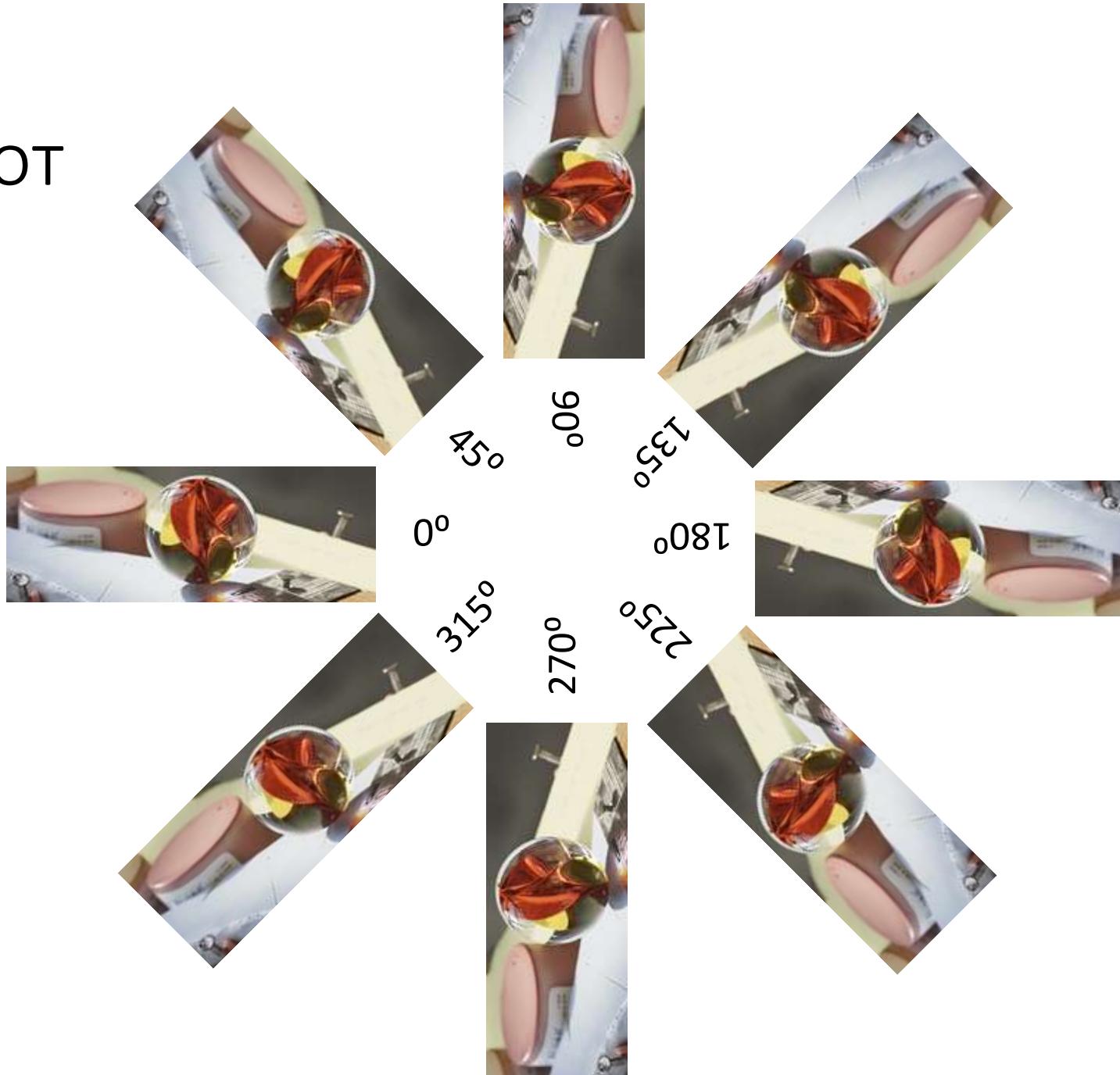
Horizontal Flip



Vertical Flip



Rotation - поворот



Rotation vs Flipping

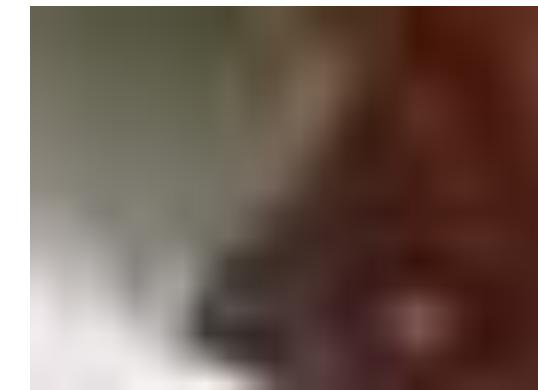
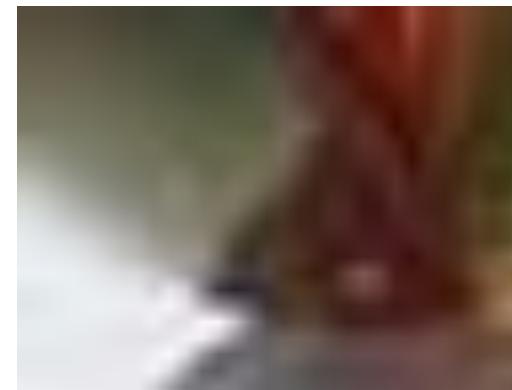
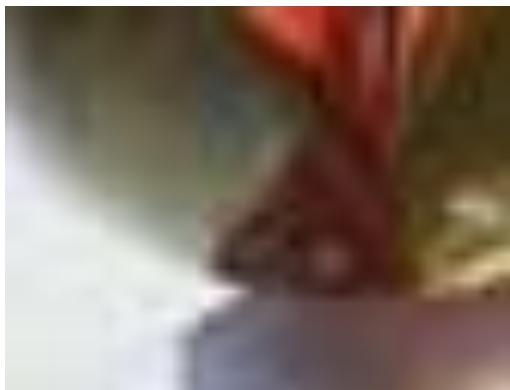
Вращение



Переворачивание



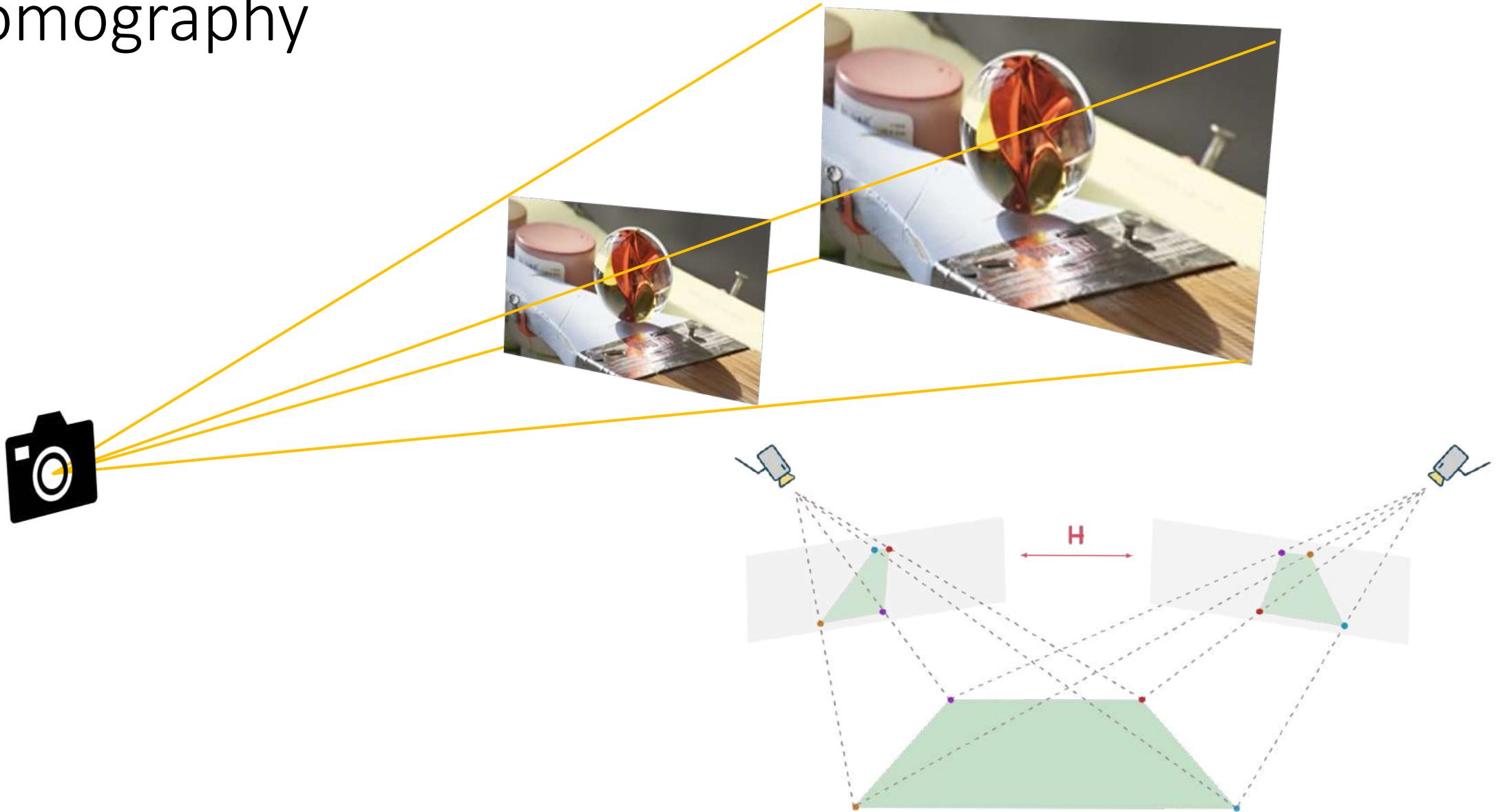
Zooming - масштабирование



Width And Height Shifting- сдвиги



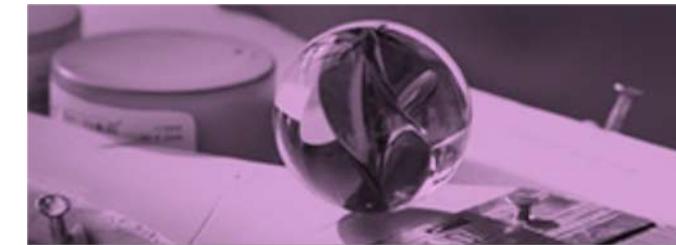
Homography



Brightness - яркость



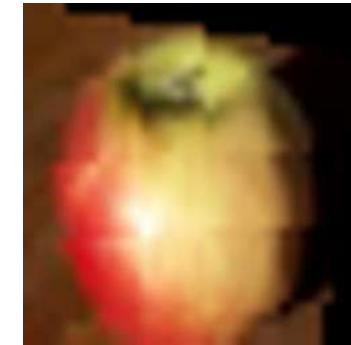
Channel Shifting – цветовые каналы



Аугментация данных



Аугментация данных позволяет разнообразить нашу выборку

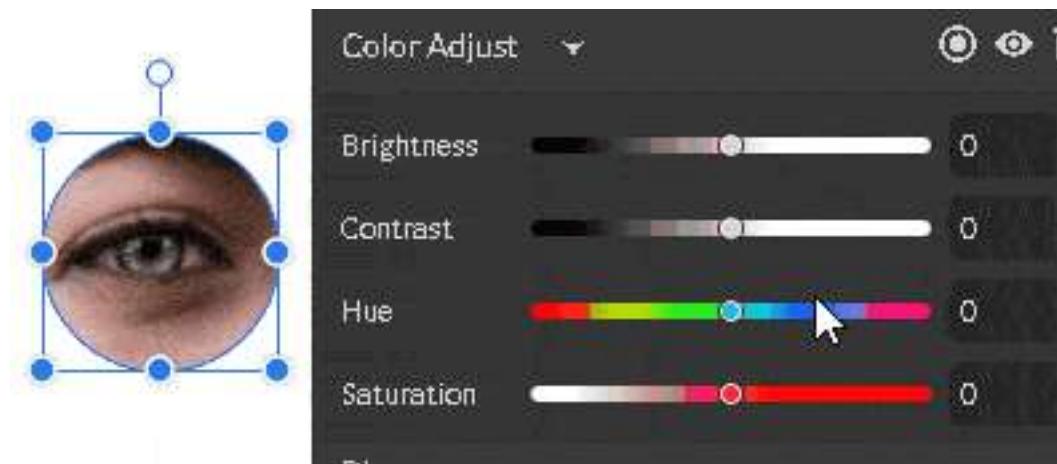


```
transform = T.Compose([
    T.ColorJitter(brightness=0.1, contrast=0.1, saturation=0.2, hue=0.0),
    T.RandomAffine(degrees=15, translate=(0.1, 0.1), scale=(0.8, 1.2),
                   shear=5),
])
```

Аугментация данных

T.ColorJitter - Случайным образом изменяет яркость, контрастность, насыщенность и оттенок изображения

- **brightness** – Насколько сильно изменять яркость
- **contrast** – Насколько сильно изменять контраст
- **saturation** – Насколько сильно изменять насыщенность
- **hue** – Как сильно изменять оттенок



Аугментация данных

T.RandomAffine - Случайное аффинное преобразование изображения, сохраняющее инвариантность центра.

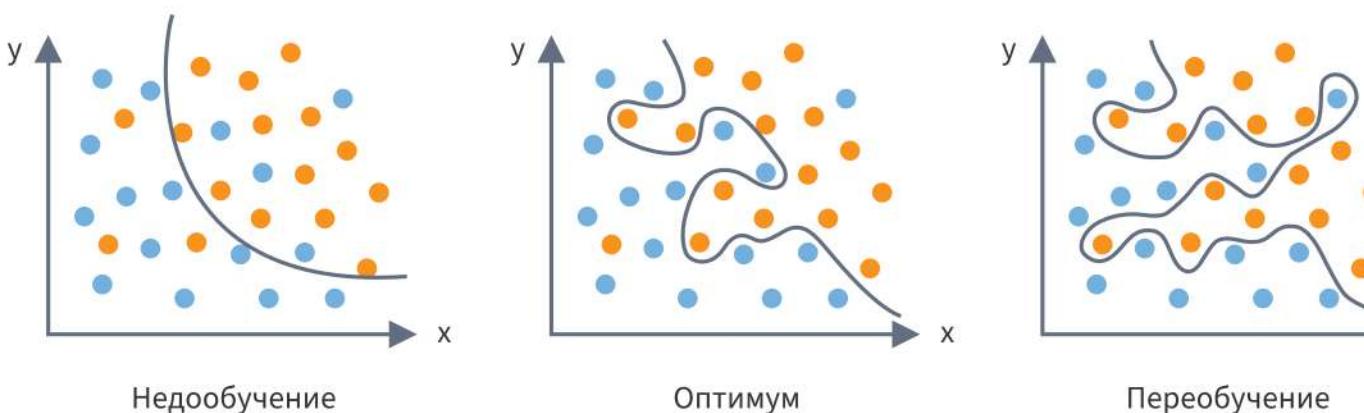
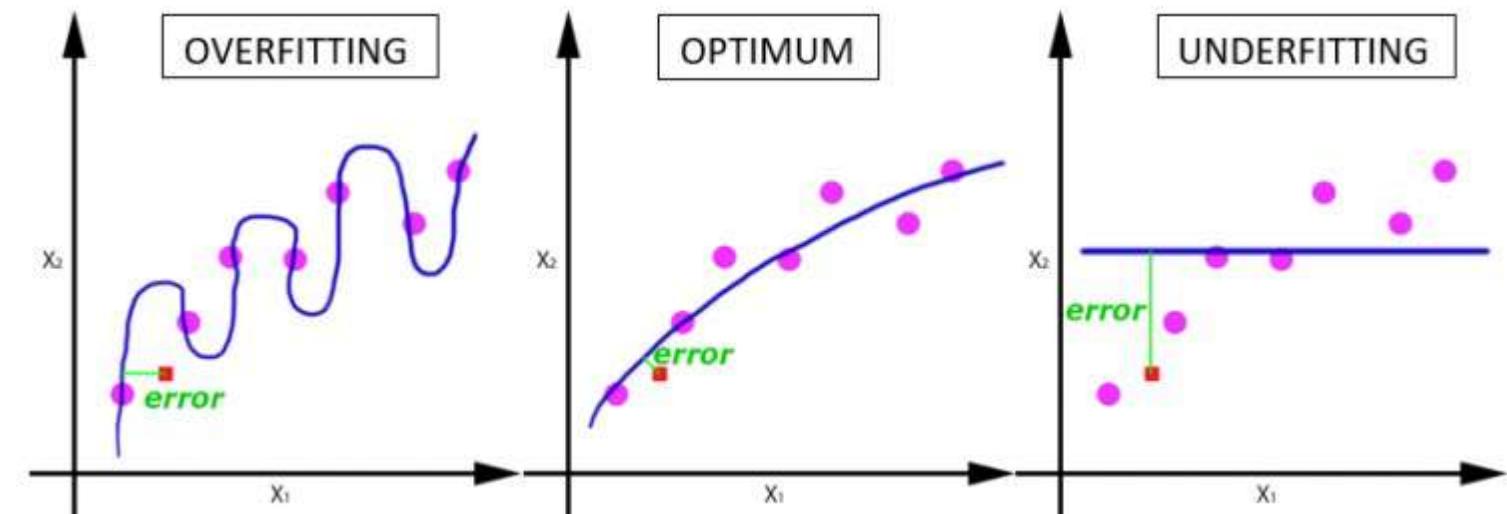
degrees – диапазон градусов для поворота

translate – кортеж для горизонтальных и вертикальных смещений

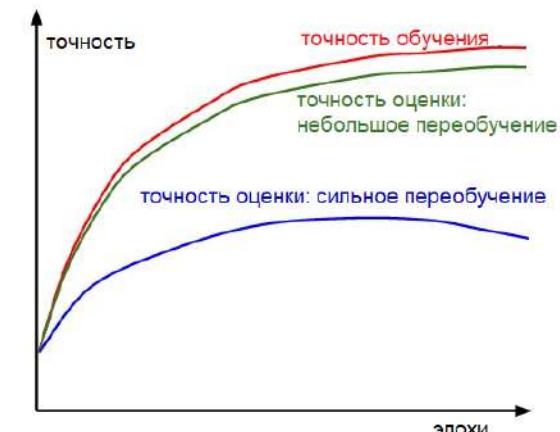
scale – интервал коэффициента масштабирования, например (a, b), затем масштаб выбирается случайным образом из диапазона $a \leq$ масштаб $\leq b$

shear – диапазон градусов на выбор. Если сдвиг является числом, то будет применен сдвиг, параллельный оси x в диапазоне (-сдвиг, +сдвиг). В противном случае, если сдвиг представляет собой последовательность из 2 значений, будет применен сдвиг, параллельный оси x в диапазоне (сдвиг [0], сдвиг[1]). В противном случае, если сдвиг представляет собой последовательность из 4 значений, будет применен сдвиг по оси x (сдвиг[0], сдвиг[1]) и сдвиг по оси y (сдвиг[2], сдвиг[3]). По умолчанию сдвиг не применяется.

Точность, переобучение

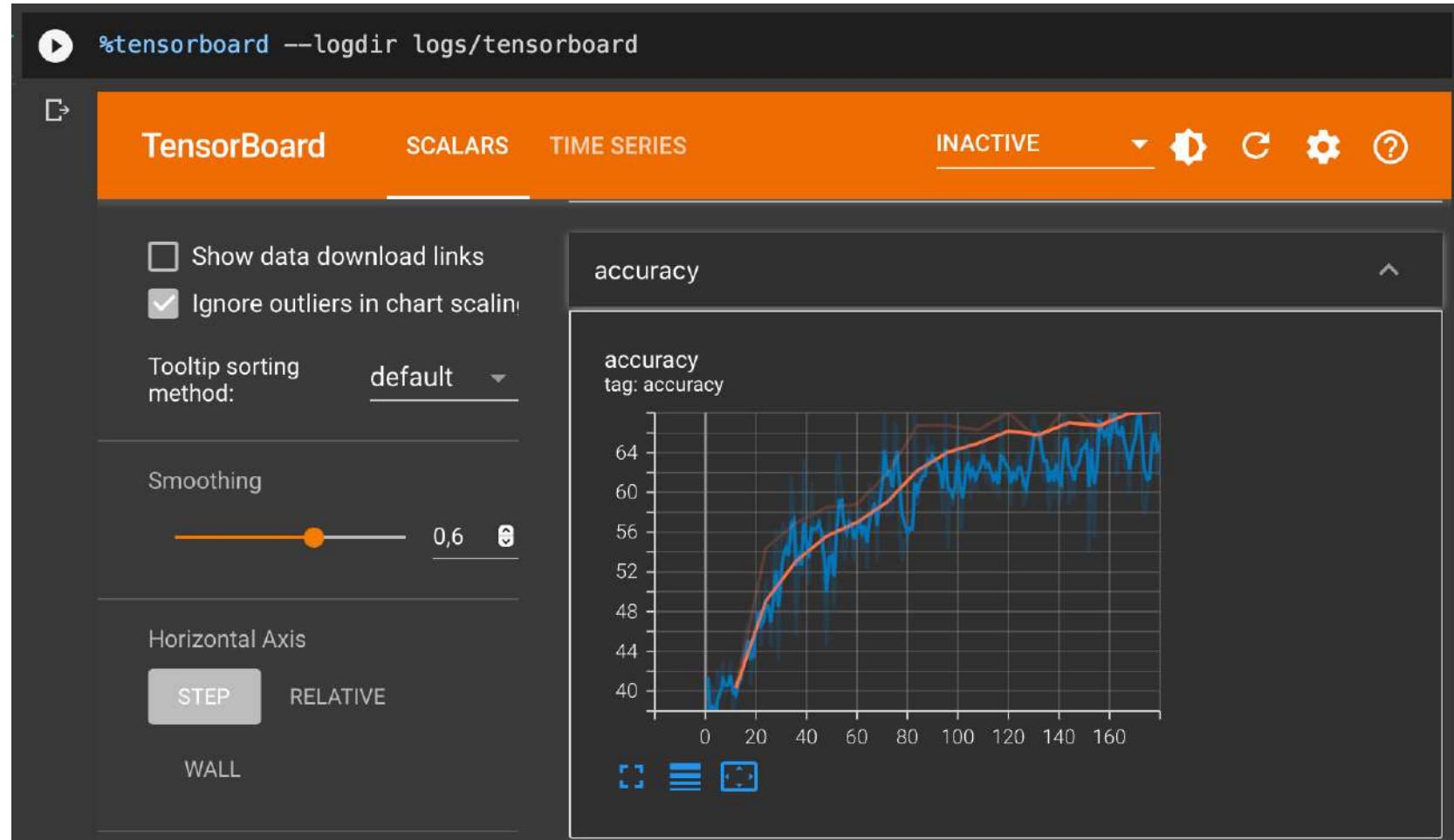


- Переобучение при долгом обучении слишком сложной модели



Tensorboard

Tensorboard - инструмент для визуализации различных параметров и переменных в процессе обучения



```
with train_summary_writer.as_default():
    tfsummary.scalar('loss', tmp[-1][0], step=pbar.n)
    tfsummary.scalar('accuracy', tmp[-1][1], step=pbar.n)
```

Регуляризация

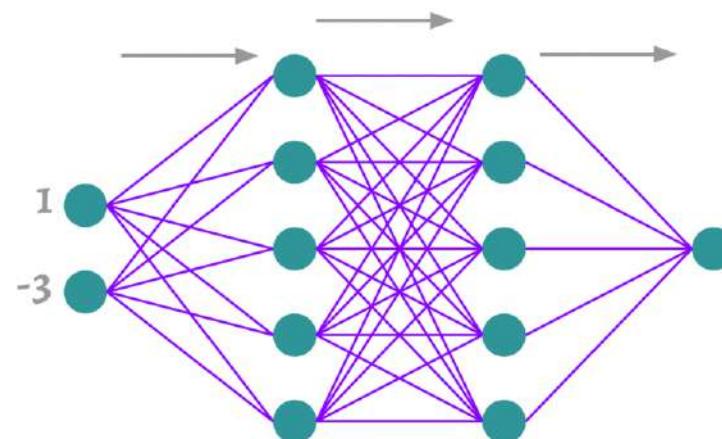
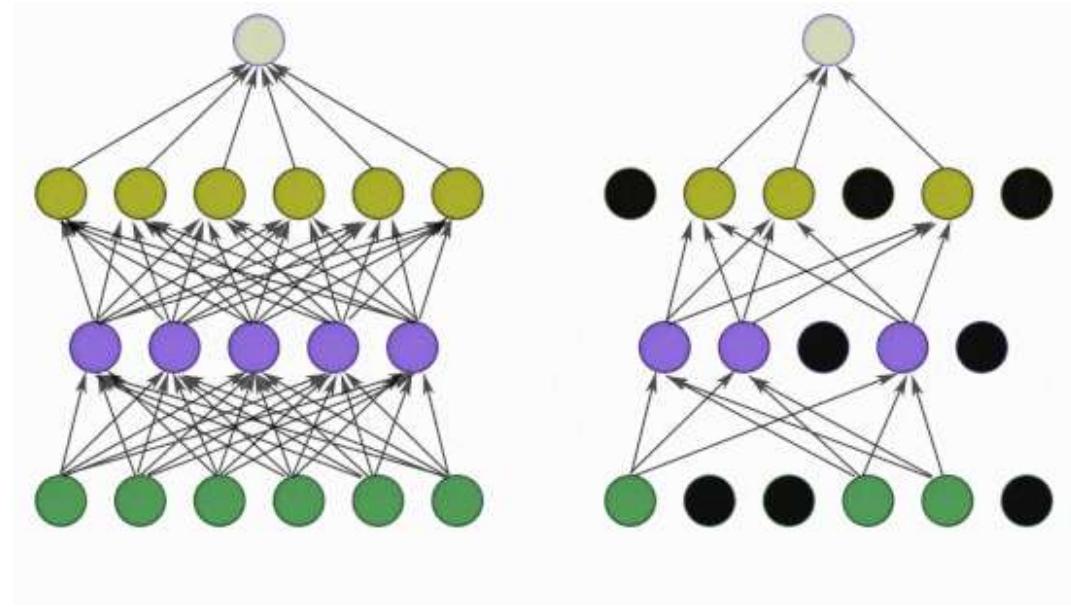
Регуляризация позволяет уменьшить эффект переобучения моделей.

Три вида регуляризации:

- Дропаут
- Штраф за сложность модели
- Label smoothing

Дропаут

- Дропаут – борьба с переобучением для уменьшения сложности модели

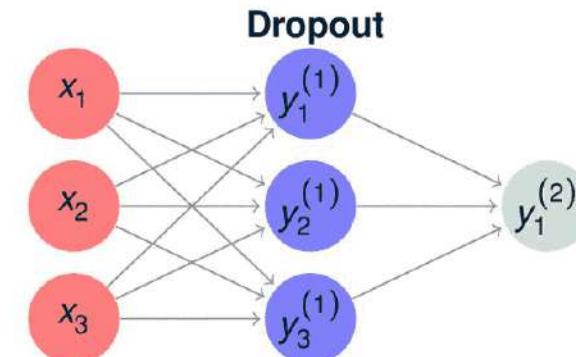


Results : [5.42, 7.89, 4.39, 5.17, 8.01, 6.27,

$$\mu = 6.19$$

$$\sigma^2 = 1.85$$

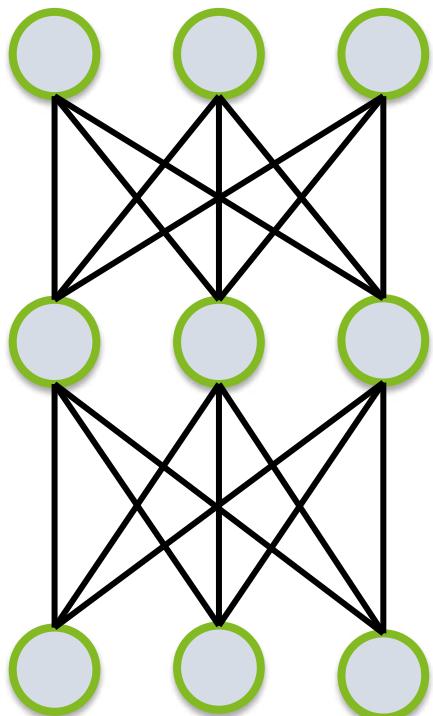
Dropout and Dropconnect



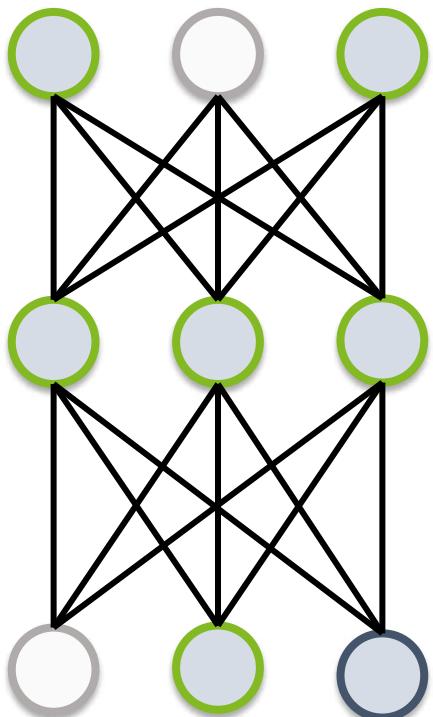
Дропаут

- Слой Dropout позволяет уменьшить эффект переобучения модели. Он достигает этого за счёт "выключения" нейронов предшествующего слоя с вероятностью p .
- Под "выключением" подразумевается зануление значений нейронов. На практике p лежит в диапазонах от 0.1 до 0.4, при этом у последующих слоёв большие значения p .
- Для того, чтобы среднее всех нейронов осталось неизменным, значения нетронутых нейронов увеличивают в $1/(1-p)$ раз. Таким образом, если отключается 50% нейронов, сигнал от остальной половины увеличивается ровно вдвое.
- Поскольку зануление случайной подвыборки нейронов происходит на каждой итерации обучения, достигается эффект "ансамбля" нейронных сетей с меньшим числом параметров.

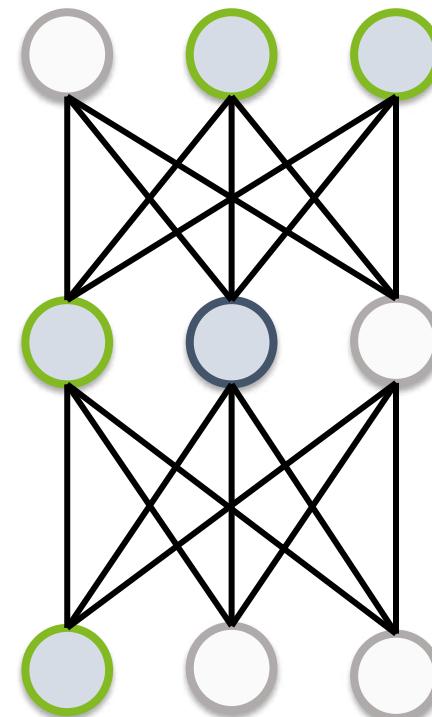
Dropout



rate = 0



rate = .2



rate = .4

Дропаут

```
nn.Conv2d(3, HIDDEN_SIZE, 3, stride=4),
```

```
nn.ReLU(),
```

```
nn.Dropout2d(p=0.2),
```

```
nn.Conv2d(HIDDEN_SIZE, HIDDEN_SIZE*2, 3, stride=1, padding=1),
```

```
nn.ReLU(),
```

```
nn.AvgPool2d(4),
```

```
nn.Dropout2d(p=0.3),
```

```
nn.Flatten(),
```

```
nn.Linear(HIDDEN_SIZE*8, classes),
```

Штраф за сложность модели

- Второй вариант регуляризации - добавление в функцию потерь второго слагаемого, штрафа за сложность модели.
- Обычно сложность модели выражается взятием нормы её параметров или весов.
- В PyTorch добавление штрафа реализовано на этапе определения оптимизатора градиентного спуска с помощью параметра **weight_decay**:

```
optimizer = optim.SGD(model.parameters(), lr=1e-3, weight_decay=1e-5)
```

- В данном примере к функции потерь при вызове метода `optimizer.step` будет добавлено второе слагаемое, равное сумме норм весов параметров, помноженное на значение параметра **weight_decay**.

$$L_1 = \sum_i (y_i - y(t_i))^2 + \lambda \sum_i |a_i|.$$

Label smoothing

Сглаживание меток заключается в изменении разметки для того, чтобы сделать модель менее уверенной в своих предсказаниях.

Рассмотрим пример для трёх классов:

- В таком случае hot-point кодировка класса 2 будет $[0, 1, 0]$.
- Если применить **label smoothing** с параметром 0.3, то сглаженное представление будет равно $[0.1, 0.8, 0.1]$.

К большому везению, данная техника автоматически встроена во многие функции потерь, в частности в `nn.CrossEntropyLoss` в качестве параметра **label_smoothing**.

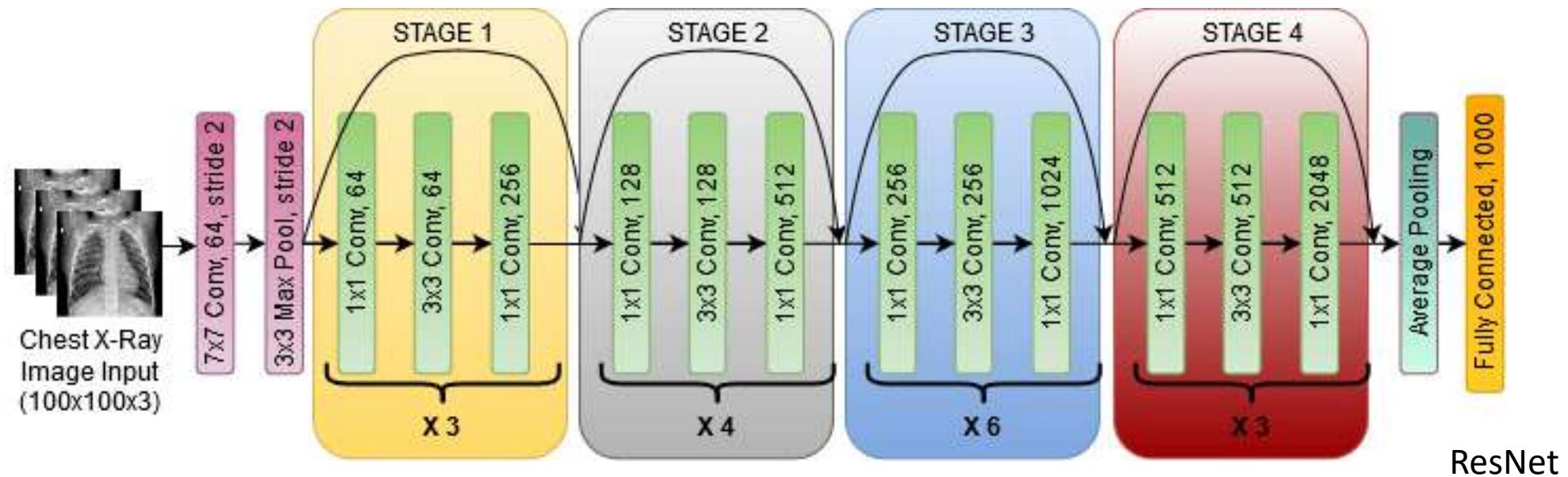
Также если в обучающей выборке содержатся ошибки в разметке, то функция потерь на сглаженных метках будет придавать им меньший вес.

Лекция 4

Перенос обучения

Разработка нейросетевых систем

Предобученные модели



```
model = torch.hub.load("pytorch-cifar-models", "cifar100_mobilnetv2_x0_5", pretrained=True)
```

Задача до

Предобученная модель распознает виды животных

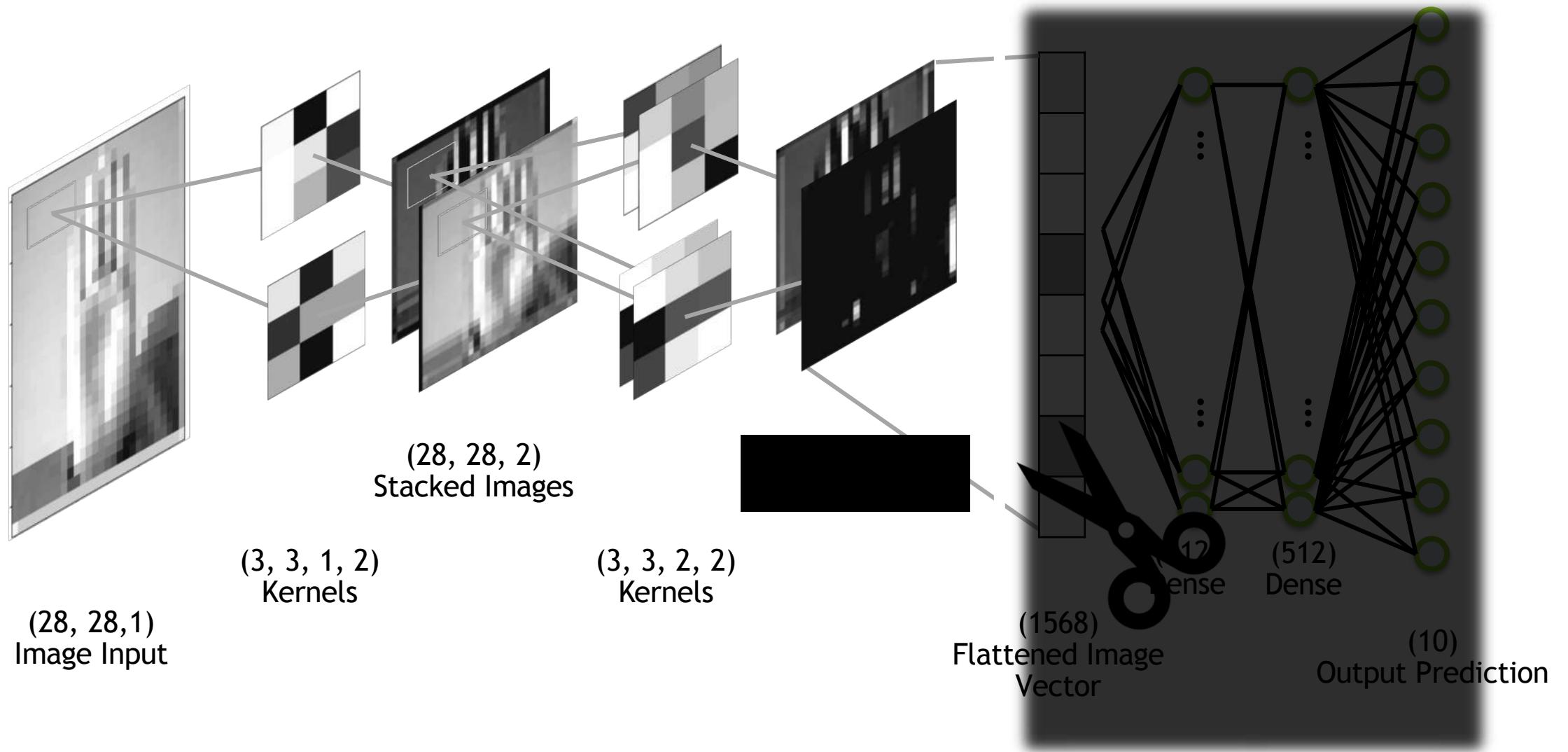


Задача после

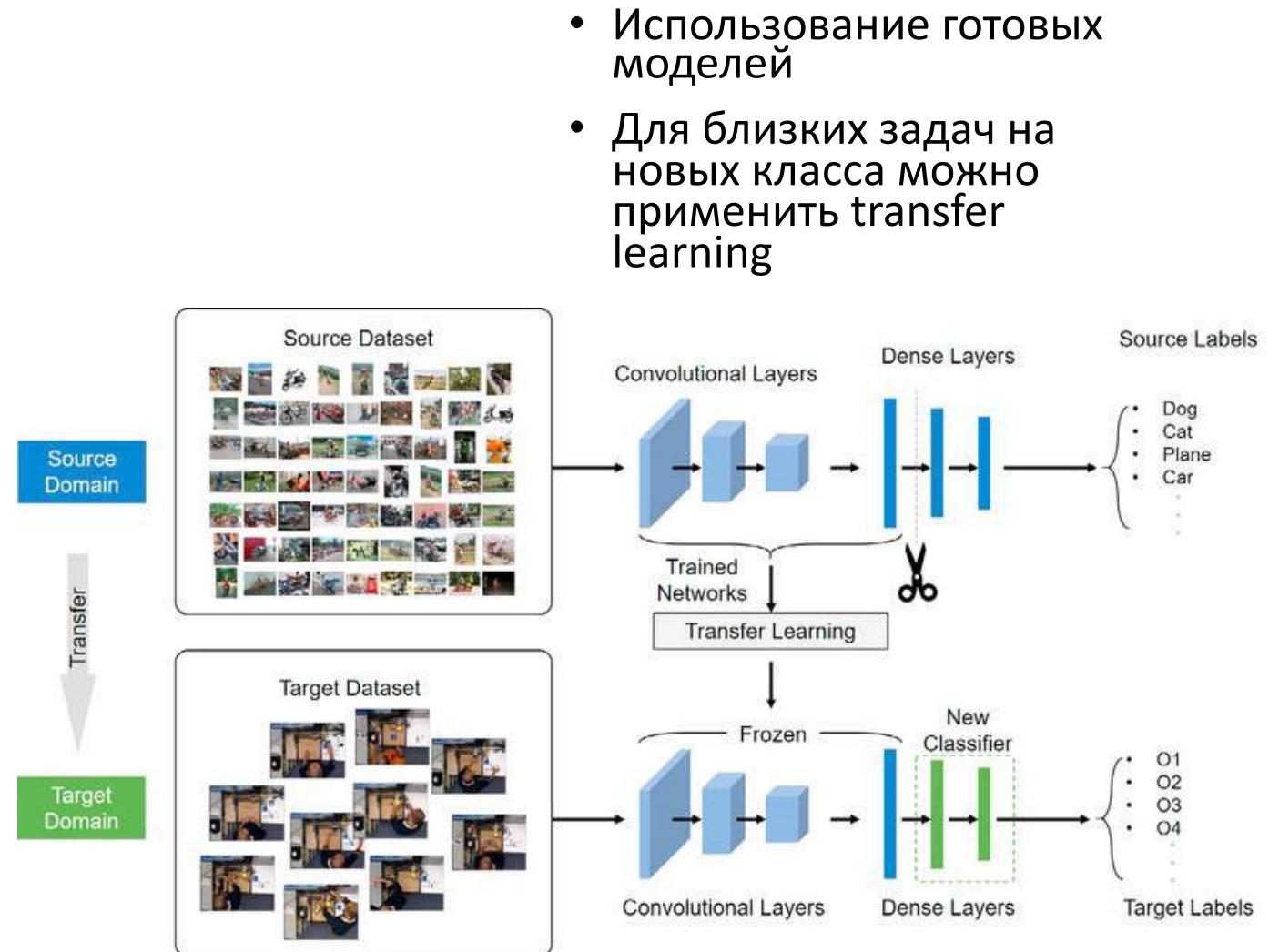
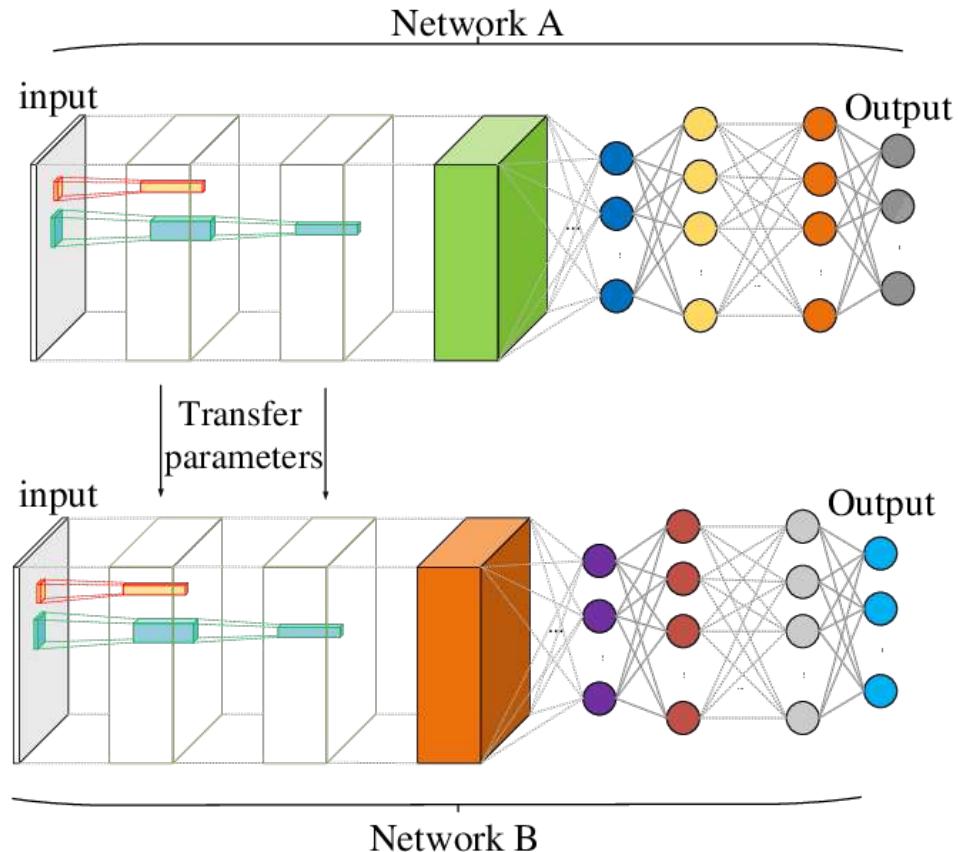
Дообученная модель распознает породы собак



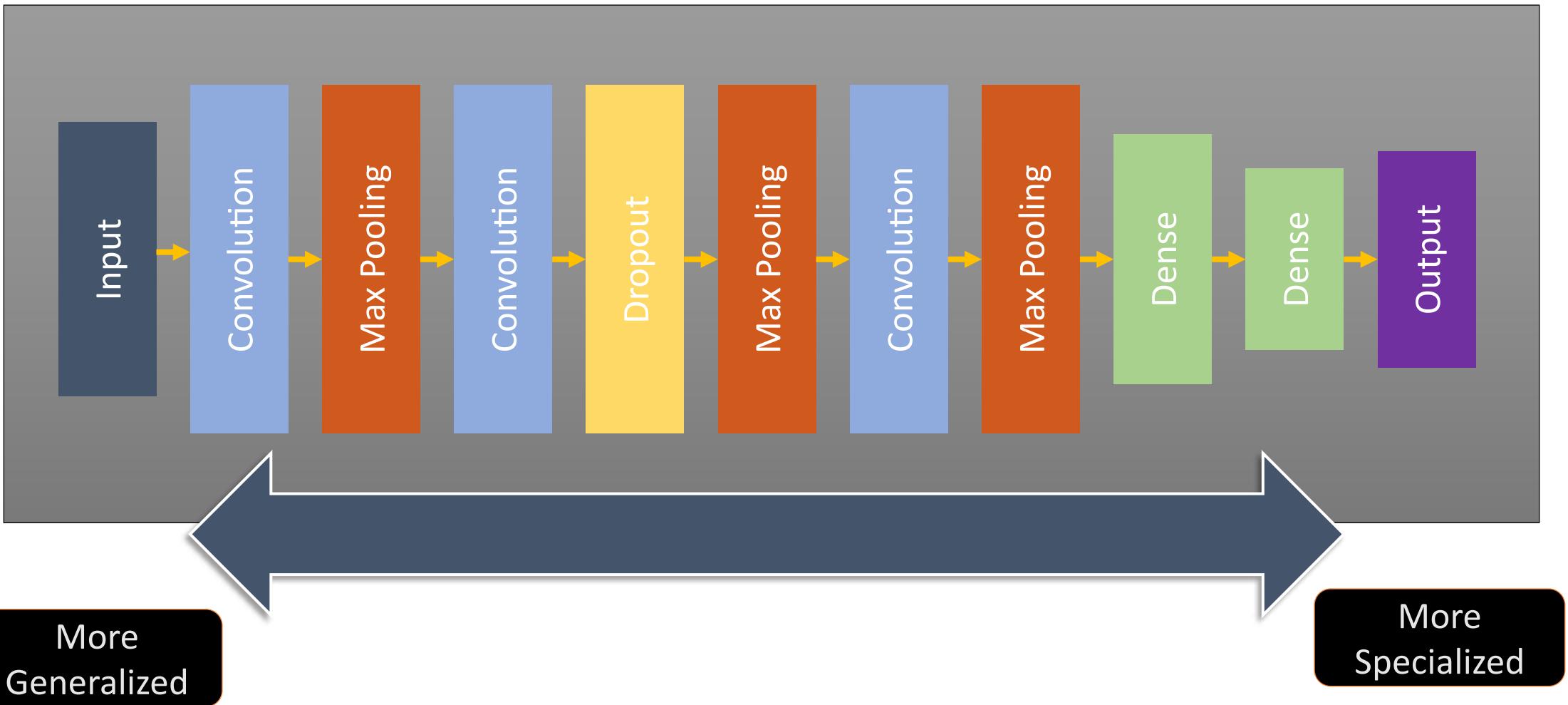
Перенос обучения



Перенос обучения - transfer learning



Transfer Learning



Заморозка модели

- После заморозки весов преобученной модели наступает этап обучения новых слоев
- Заморозка нужна чтобы не испортить уже обученные веса

```
for i, (name, param) in enumerate(new_model.named_parameters()):  
    if i < total - keep_last:  
        param.requires_grad = False  
    else:  
        params_to_update.append(param)  
        print("\t", name)  
        param.requires_grad = True
```



Обучаемые параметры:

1.fc.weight
1.fc.bias

Total params: 272,019

Trainable params: 195

Non-trainable params: 271,824

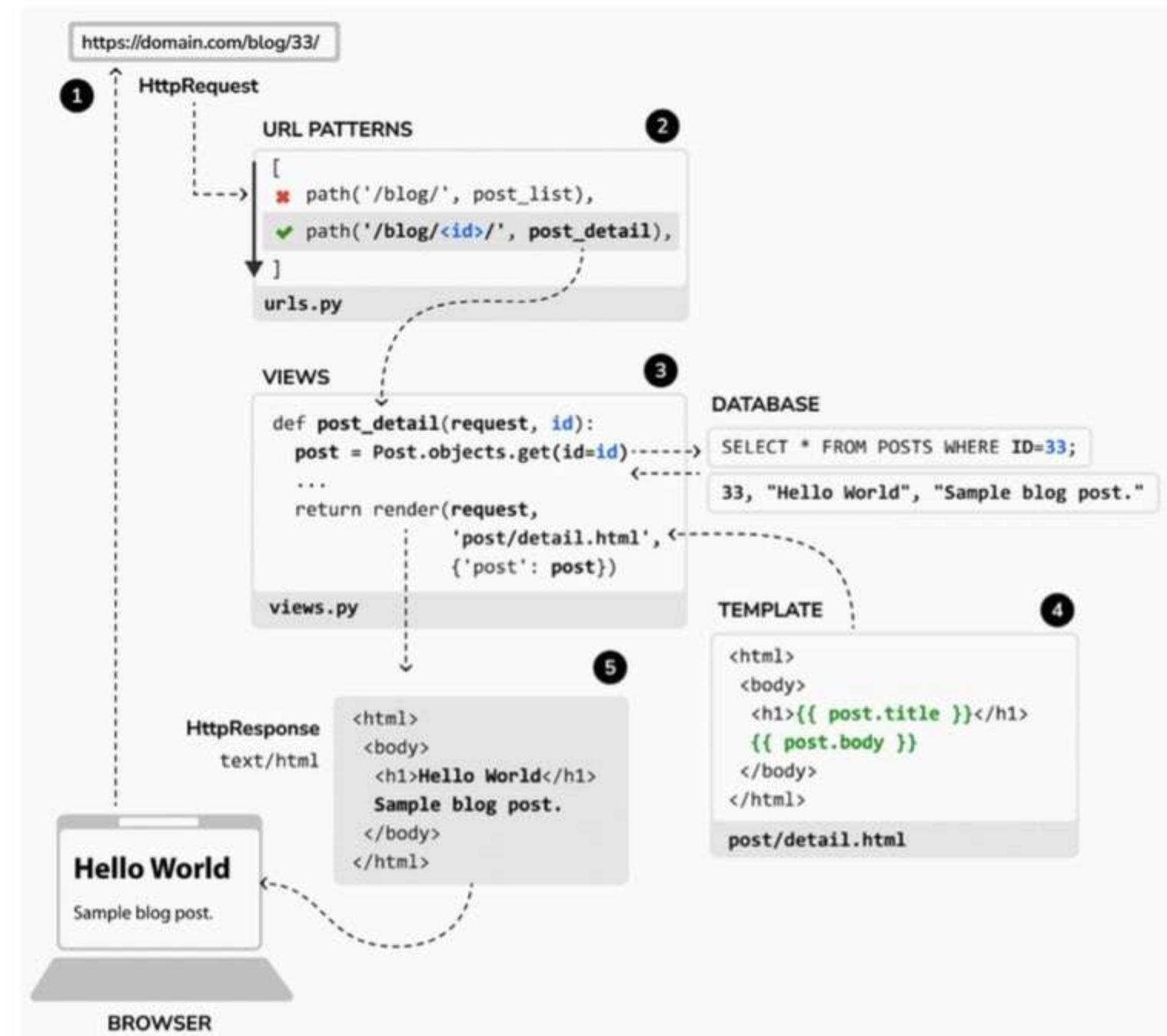
Fine Tuning- дообучение



- Когда последние слои не обучены, они вносят большую ошибку и через обратное распространение влияют на первые
- После обучения выходных слоев все остальные веса можно разморозить и продолжить обучение
- Это позволяет тонко настроить все веса под нашу задачу и еще больше повысить точность

Django

- Django – это MVC фреймворк
- При обработке запроса сначала обрабатывается URL
- Решается, какой view будет его обрабатывать
- View обращается к БД или нейросети
- Результаты вносятся в шаблон Template, получается HTML



Стохастический градиентный спуск

$L(f(\mathbf{x}(i); \boldsymbol{\theta}), y(i))$ – значение функции потерь

$f(\mathbf{x}(i); \boldsymbol{\theta})$ – результат вычисления нейронной сети от входа $\mathbf{x}(i)$ и параметров (весов) $\boldsymbol{\theta}$

Обновление на k -ой итерации стохастического градиентного спуска (СГС)

Require: скорость обучения ϵ_k

Require: Начальные значения параметров $\boldsymbol{\theta}$

while критерий остановки не выполнен **do**

Выбрать из обучающего набора мини-батч m примеров $\{\mathbf{x}(1), \dots, \mathbf{x}(m)\}$ и соответствующие им метки $y(i)$.

Вычислить оценку градиента: $g \leftarrow + (1/m) \nabla_{\boldsymbol{\theta}} \sum_i L(f(\mathbf{x}(i); \boldsymbol{\theta}), y(i))$.

Применить обновление: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon g$.

end while

Adagrad

Алгоритм AdaGrad по отдельности адаптирует скорости обучения всех параметров модели. Для параметров, по которым частная производная функции потерь наибольшая, скорость обучения уменьшается быстро, а если частная производная мала, то и скорость обучения уменьшается медленнее. В итоге больший прогресс получается в направлениях пространства параметров со сравнительно пологими склонами

Алгоритм AdaGrad

Require: глобальная скорость обучения ϵ Require: начальные значения параметров θ
Require: небольшая константа δ , например 10^{-7} , для обеспечения численной устойчивости.

Инициализировать переменную для агрегирования градиента $r=0$

while критерий остановки не выполнен do

Выбрать из обучающего набора мини-пакет m примеров $\{x(1), \dots, x(m)\}$ и
соответствующие им метки $y(i)$.

Вычислить градиент: $g \leftarrow (1/m) \nabla_{\theta} \sum_i L(f(x(i); \theta), y(i))$.

Агрегировать квадраты градиента: $r \leftarrow r + g \odot g$.

Вычислить обновление: $\Delta\theta \leftarrow -\epsilon / (\delta + \sqrt{r}) \odot g$

Применить обновление: $\theta \leftarrow \theta + \Delta\theta$. end while

RMSProp

AdaGrad уменьшает скорость обучения, принимая во внимание всю историю квадрата градиента, и может случиться так, что скорость станет слишком малой еще до достижения такой выпуклой структуры.

В алгоритме RMSProp используется экспоненциально затухающее среднее, т. е. далекое прошлое отбрасывается

Require: глобальная скорость обучения ϵ , скорость затухания ρ **Require:** начальные значения параметров θ

Require: небольшая константа δ , например 10^{-6} , для стабилизации деления на малые числа

Инициализировать переменную для агрегирования градиента $r = 0$

while критерий остановки не выполнен **do**

Выбрать из обучающего набора мини-пакет m примеров $\{x(1), \dots, x(m)\}$ и соответствующие им метки y_i .

Вычислить градиент: $g \leftarrow (1/m) \nabla_{\theta} \sum_i L(f(x(i); \theta), y(i))$.

Агрегировать квадраты градиента: $r \leftarrow \rho r + (1 - \rho) g \odot g$.

Вычислить обновление параметров: $\Delta \theta \leftarrow -\epsilon / \sqrt{\delta + r} \odot g$

Применить обновление: $\theta \leftarrow \theta + \Delta \theta$.

end while

Импульсный метод

Стохастический градиентный спуск (СГС) с учетом импульса

Require: скорость обучения ε , параметр импульса α

Require: начальные значения параметров θ , начальная скорость v

while критерий остановки не выполнен **do**

Выбрать из обучающего набора мини-пакет m примеров $\{x(1), \dots, x(m)\}$ и соответствующие им метки $y(i)$.

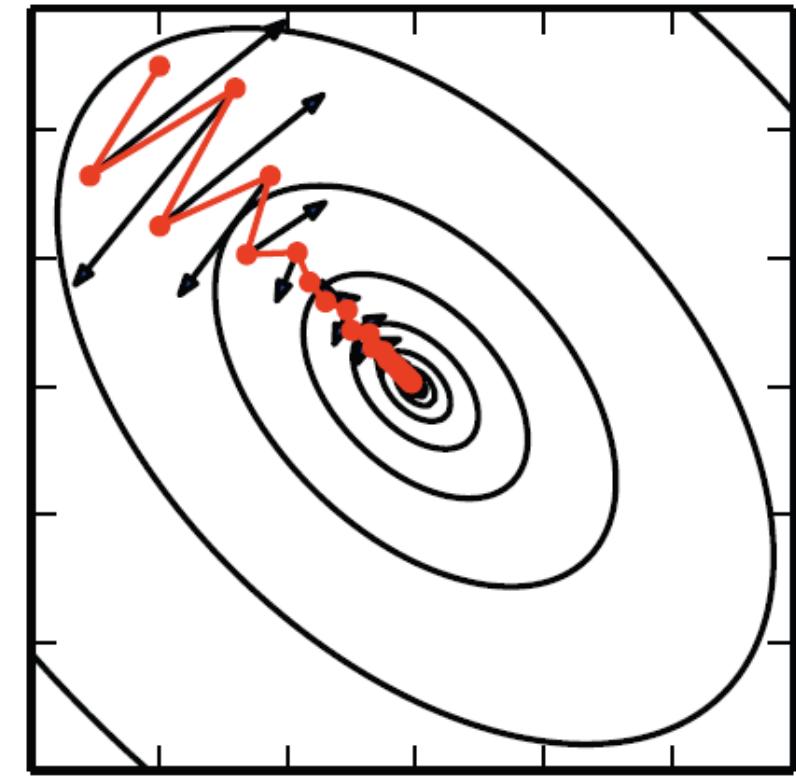
Вычислить оценку градиента:

$$g \leftarrow (1/m) \nabla_{\theta} \sum_i L(f(x(i); \theta), y(i)).$$

Вычислить обновление скорости: $v \leftarrow \alpha v - \varepsilon g$.

Применить обновление: $\theta \leftarrow \theta + v$.

end while



Импульсный алгоритм можно рассматривать как имитацию движения частицы, подчиняющейся динамике Ньютона.

Adam

«Adam» – сокращение от «adaptive moments» (адаптивные моменты). Его правильнее всего рассматривать как комбинацию RMSProp и импульсного метода

Require: величина шага ϵ (по умолчанию 0.001).

Require: коэффициенты экспоненциального затухания для оценок моментов ρ и ρ' , принадлежащие диапазону $[0, 1]$ (по умолчанию 0.9 и 0.999 соответственно).

Require: небольшая константа δ для обеспечения численной устойчивости (по умолчанию 10^{-8}).

Require: начальные значения параметров θ .

Инициализировать переменные для первого и второго моментов $s = \mathbf{0}, r = \mathbf{0}$

Инициализировать шаг по времени $t = 0$

while критерий остановки не выполнен **do**

Выбрать из обучающего набора мини-пакет m примеров $\{x(1), \dots, x(m)\}$ и соответствующие им метки y_i .

Вычислить градиент: $\mathbf{g} \leftarrow (1/m) \nabla \theta \sum_i L(f(x(i); \theta), y(i))$.

$t \leftarrow t + 1$

Обновить смещенную оценку первого момента: $s \leftarrow \rho_1 s + (1 - \rho_1) \mathbf{g}$

Обновить смещенную оценку второго момента: $r \leftarrow \rho_2 r + (1 - \rho_2) \mathbf{g} \odot \mathbf{g}$

Скорректировать смещение первого момента: $s \leftarrow s / (1 - \rho_t)$

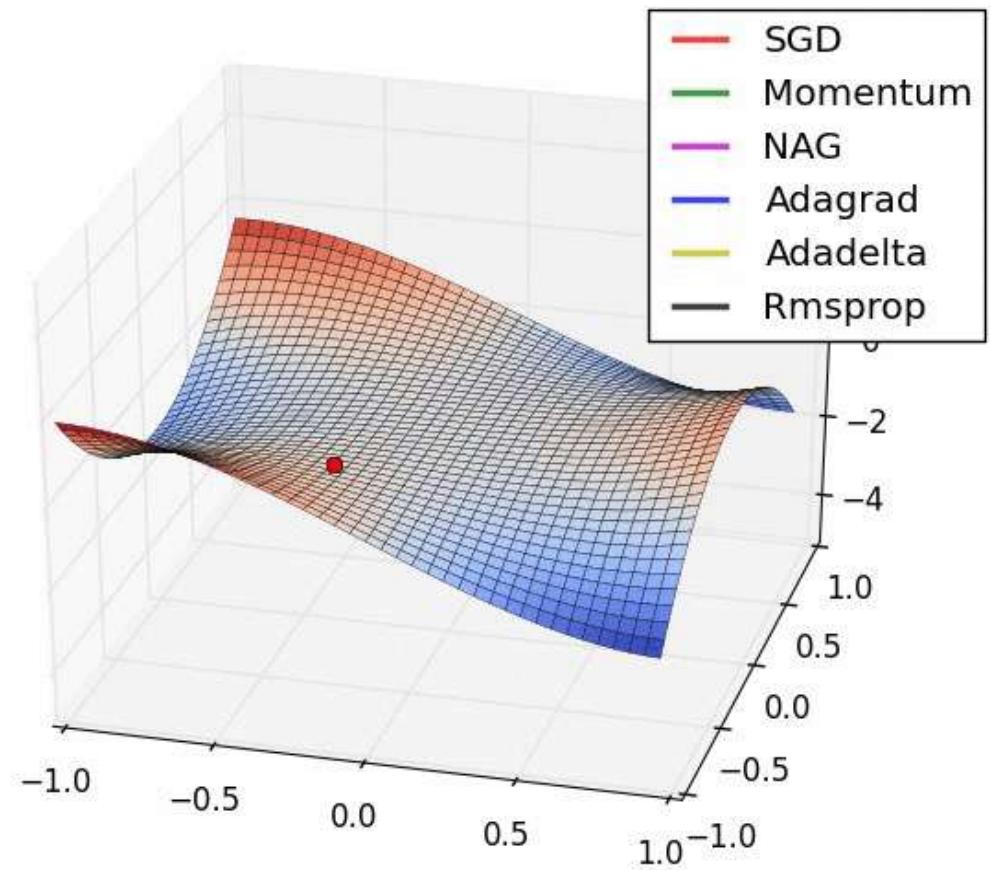
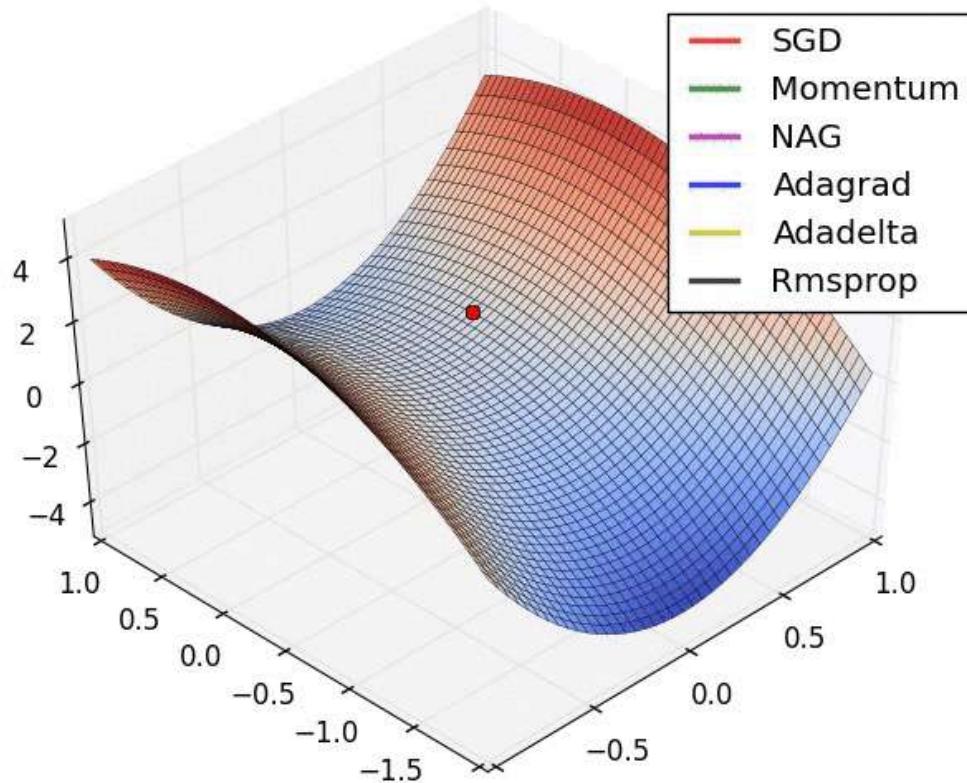
Скорректировать смещение второго момента: $r \leftarrow r / (1 - \rho_t)$

Вычислить обновление: $\delta\theta = -\epsilon s / \sqrt{(s+r)}$

Применить обновление: $\theta \leftarrow \theta + \Delta\theta$.

end while

Другие оптимизаторы



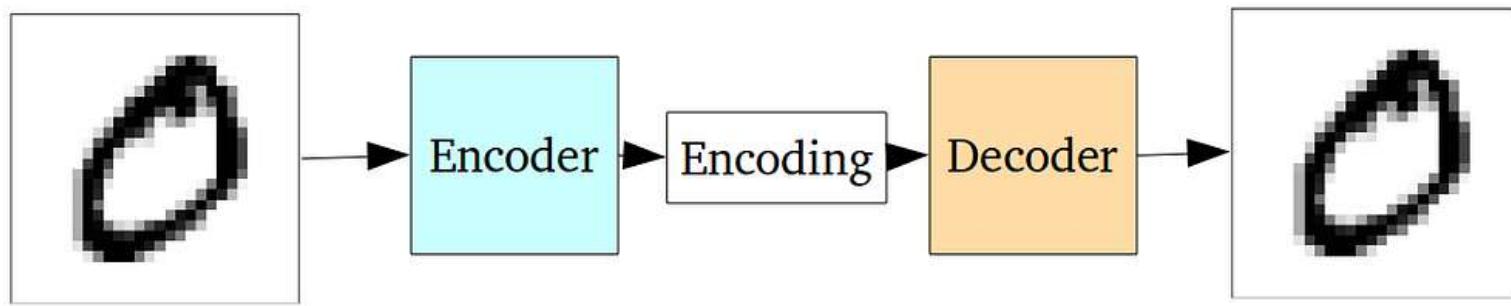
Лекция 5

Генеративные сети

Разработка нейросетевых систем

Автоэнкодер

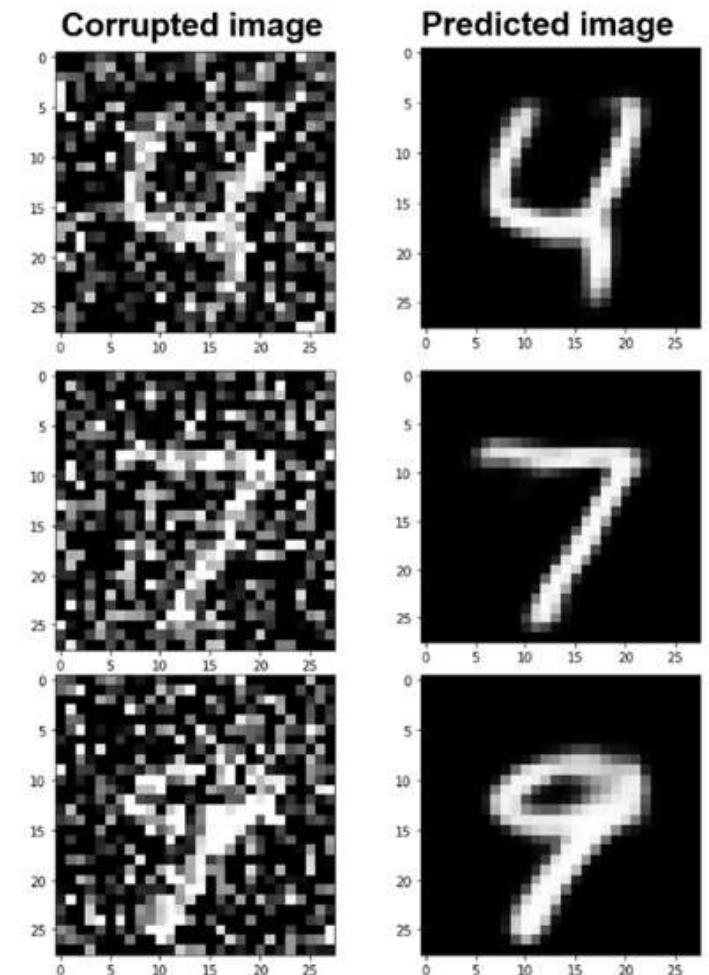
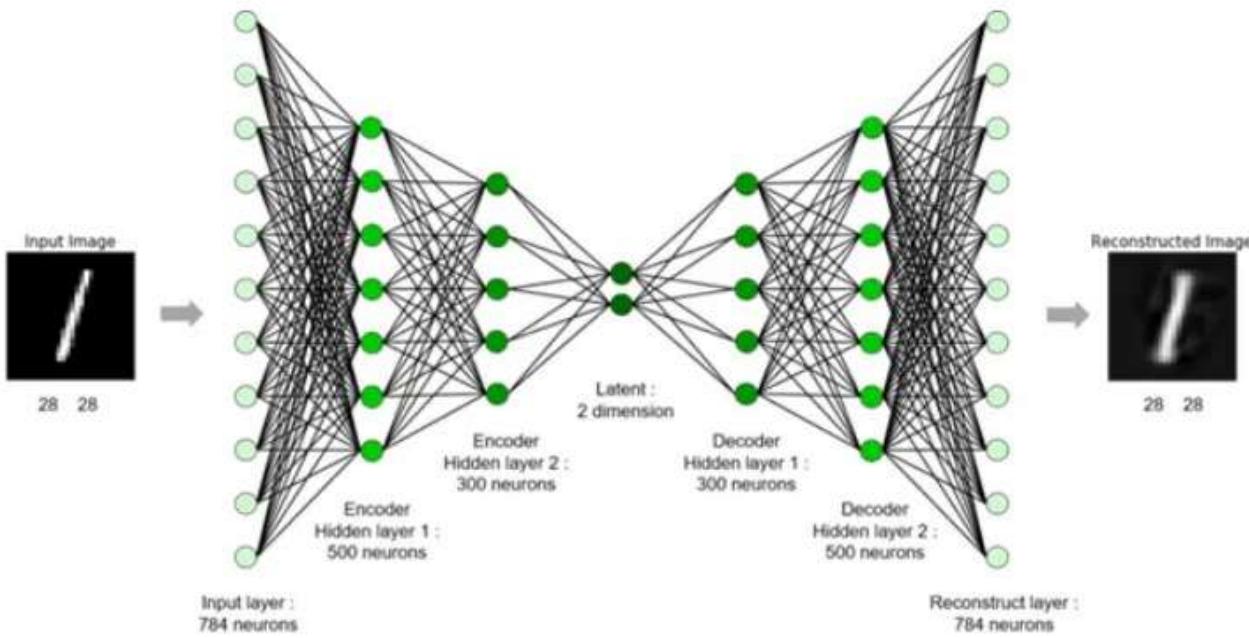
- Автоэнкодеры появились в начале 90-ых годов. Это нелинейное обобщение метода главных компонент.
- Энкодер генерирует закодированные данные таким образом, чтобы они были пригодны для восстановления входных данных



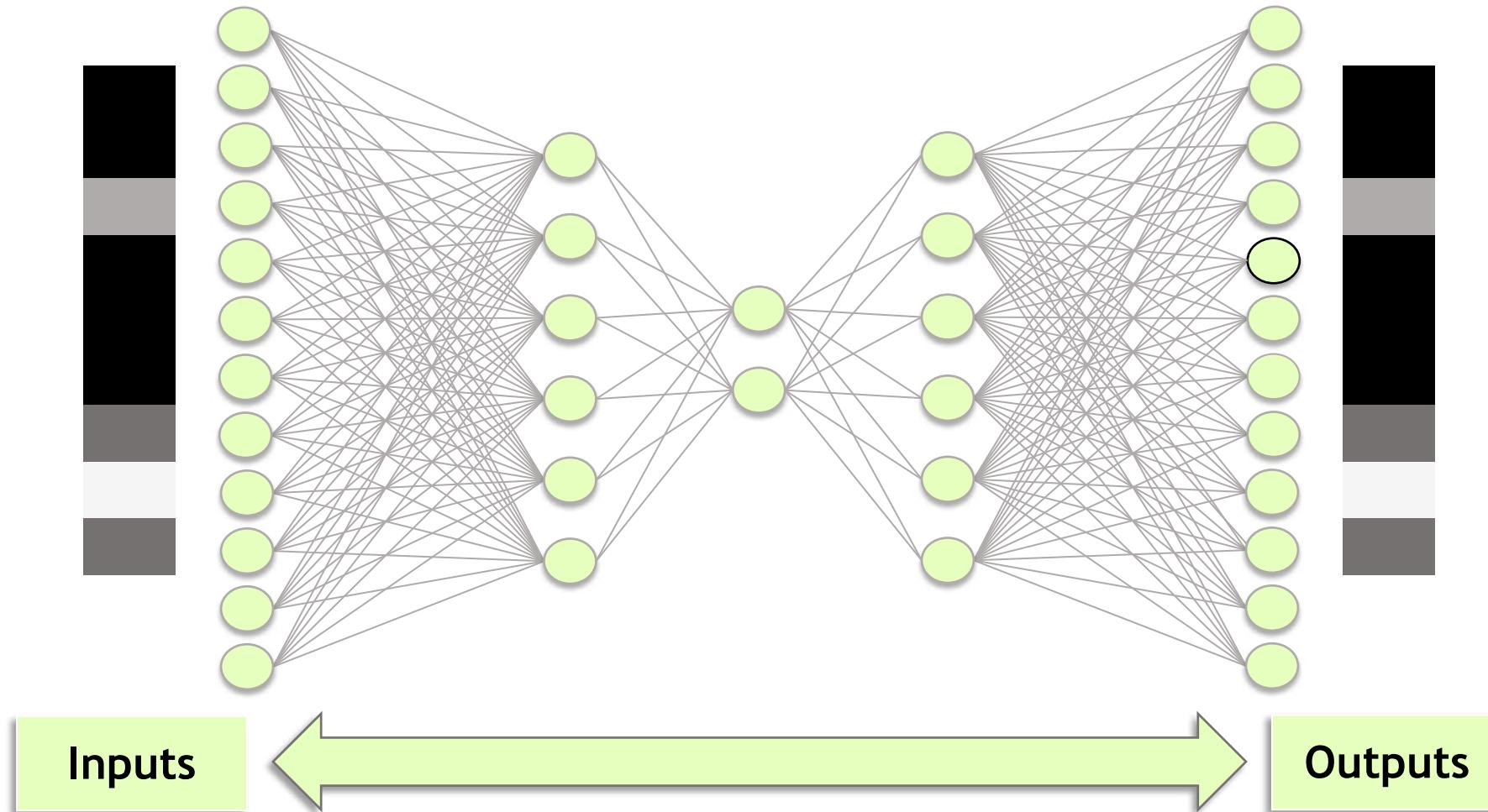
- Функция потерь выбирается как среднеквадратичная ошибка или как кросс-энтропия между входными и выходными данными
- Она не позволяет нейросети создавать выходные данные, сильно отличающиеся от входных

Denosing Autoencoder

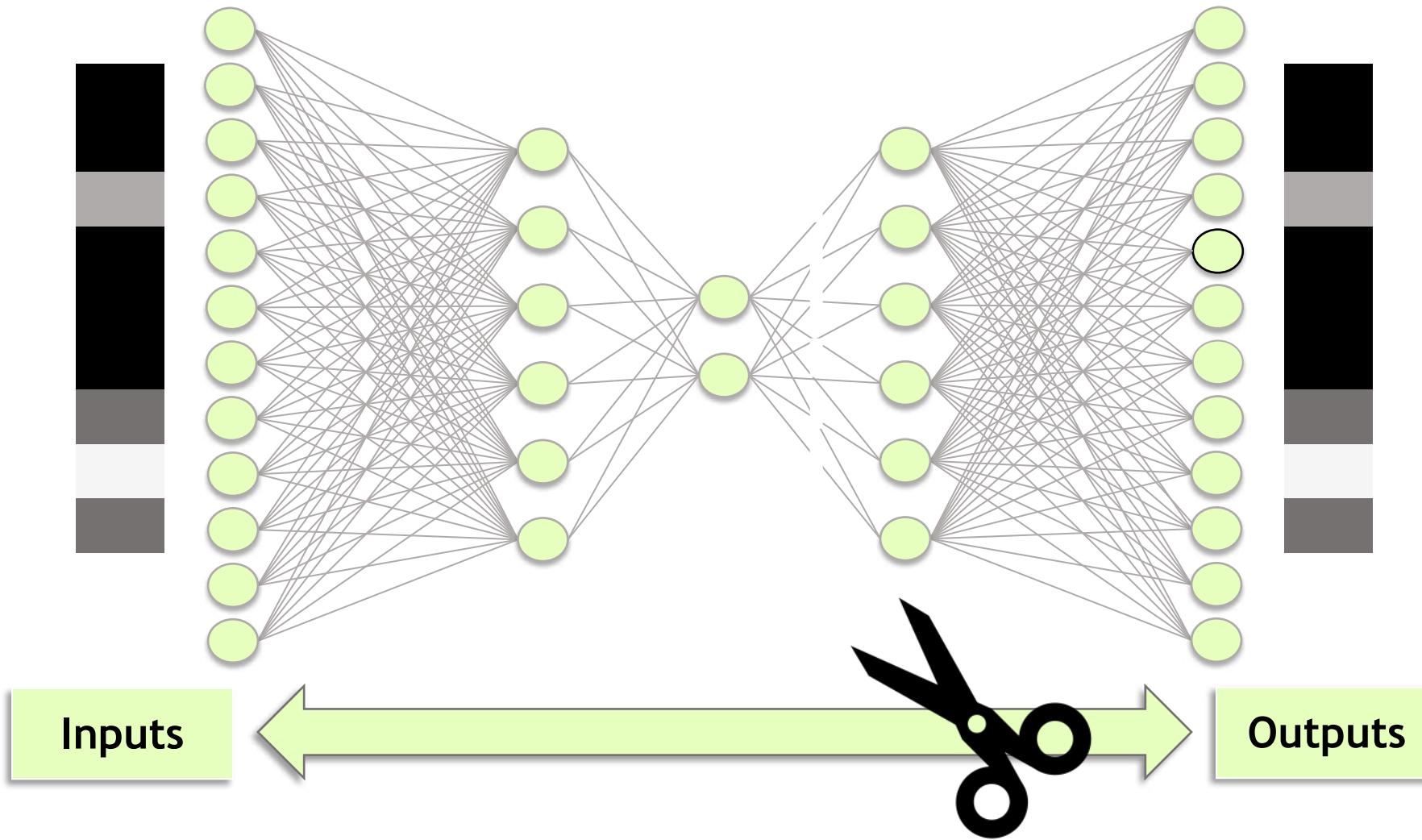
- Обучаем модель на восстановлении изображения
- Используем для удаления шума из изображения



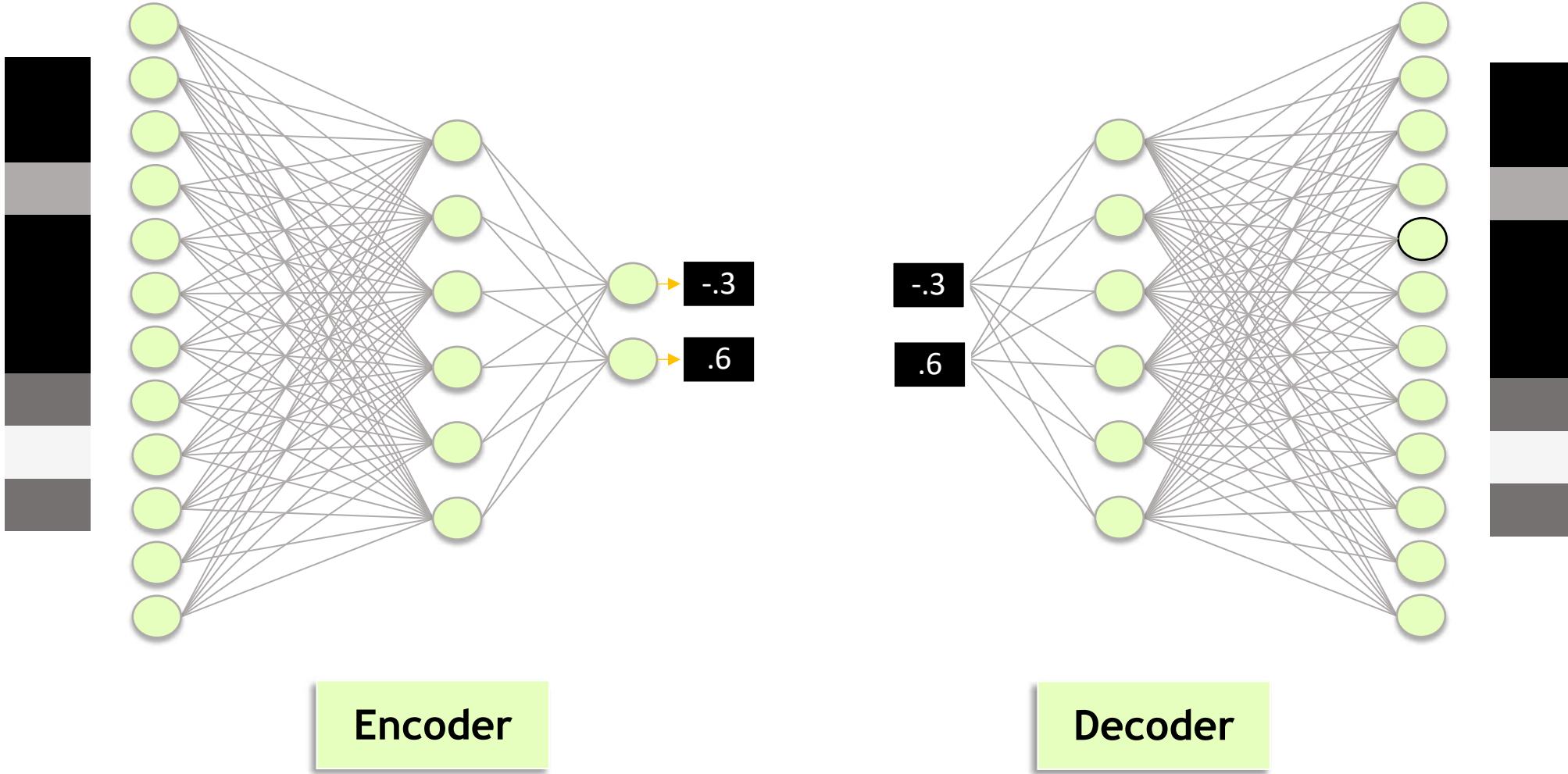
Autoencoders



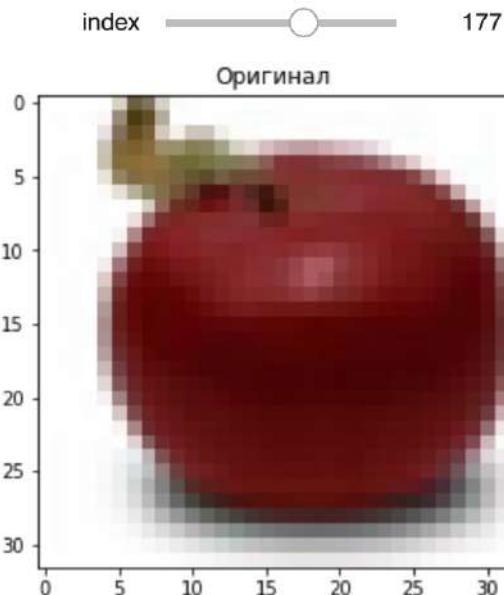
Обучение двух моделей



Latent space

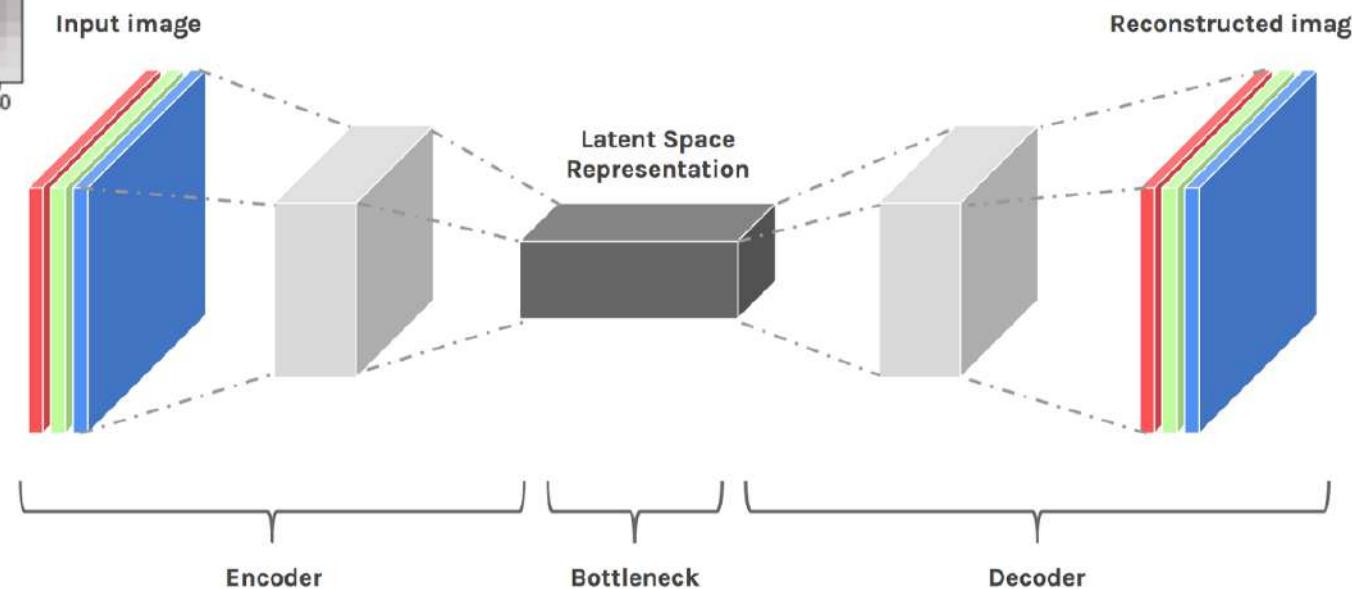


Автоэнкодер для Cifar



- Воспользуемся Cifar100 чтобы обучить автоэнкодер
- Здесь мы не используем метку label – у нас фактически неразмеченный набор данных

```
Cifar100_AE(  
    (norm): Normalize()  
    (encoder): Sequential(  
        (0): Linear(in_features=3072, out_features=512, bias=True)  
        (1): ELU(alpha=1.0)  
        (2): Linear(in_features=512, out_features=256, bias=True)  
        (3): ELU(alpha=1.0)  
        (4): Linear(in_features=256, out_features=64, bias=True)  
        (5): Tanh()  
    )  
    (decoder): Sequential(  
        (0): Linear(in_features=64, out_features=256, bias=True)  
        (1): ELU(alpha=1.0)  
        (2): Linear(in_features=256, out_features=512, bias=True)  
        (3): ELU(alpha=1.0)  
        (4): Linear(in_features=512, out_features=3072, bias=True)  
    )  
)
```



Автоэнкодер для Cifar

```
class Cifar100_AE(nn.Module):
    def __init__(self, hidden_size=32, classes=100):
        super(Cifar100_AE, self).__init__()
        # https://blog.jovian.ai/image-classification-of-cifar100-dataset-us
        self.norm = Normalize([0.5074, 0.4867, 0.4411], [0.2011, 0.1987, 0.2025])
        self.encoder = nn.Sequential(
            nn.Linear(32*32*3, hidden_size),
            nn.ELU(),
            nn.Linear(hidden_size, hidden_size//2),
            nn.ELU(),
            nn.Linear(hidden_size//2, hidden_size//8),
            nn.Tanh()
        )
        self.decoder = nn.Sequential(
            nn.Linear(hidden_size//8, hidden_size//2),
            nn.ELU(),
            nn.Linear(hidden_size//2, hidden_size),
            nn.ELU(),
            nn.Linear(hidden_size, 32*32*3),
        )
```

- Автоэнкодер состоит из 2-ух частей, все слои сгруппированы в эти части
- Это позволяет при предсказании получить больше ответов: выход энкодера и декодера

```
def forward(self, input):
    normed = self.norm(input)
    encoded = self.encoder(normed)
    out = self.decoder(encoded)
    return out, encoded, normed
```

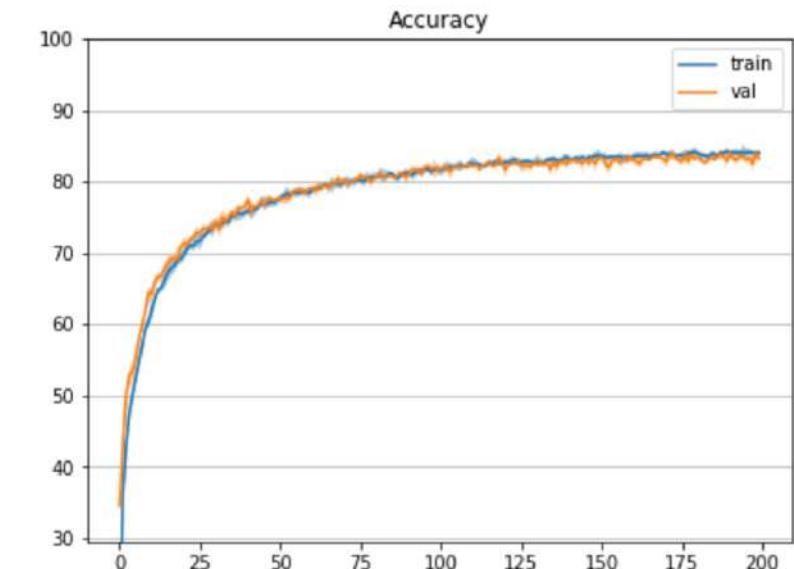
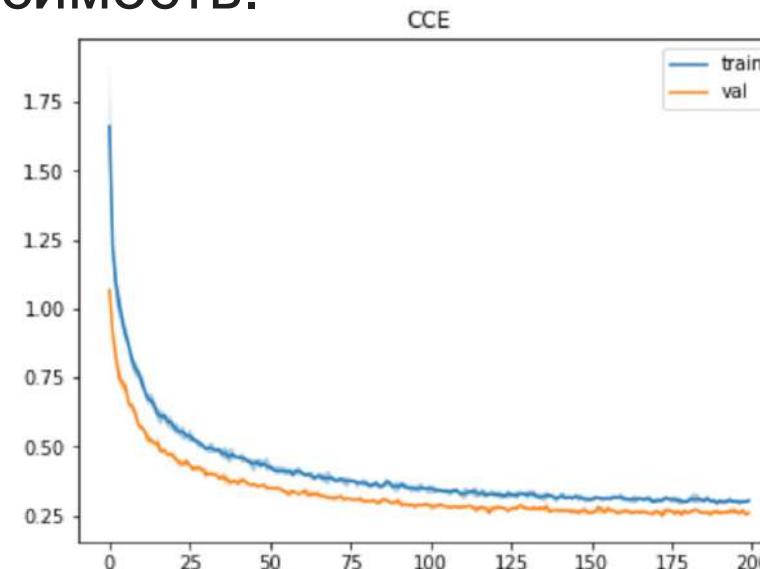
Коэффициент детерминации

- Accuracy – здесь мы считаем не процент правильных ответов, а коэффициент детерминации

$$R^2 = 1 - \frac{D[y|x]}{D[y]} = 1 - \frac{\sigma^2}{\sigma_y^2}$$

- Коэффициент детерминации для модели с константой принимает значения от - бесконечность до 1. Чем ближе значение коэффициента к 1, тем сильнее зависимость.

- Функция потерь – MSE



Эмбеддинги

- Embedding – векторное представление
- Используем выход энкодера в качестве координат для визуализации
- Сделаем проекцию на плоскость с помощью метода главных компонент РСА

```
# прямой + обратный проходы + оптимизация
out, embedding, norm = model(inputs)
embeddings.append(embedding.detach().cpu().numpy())
images.append(inputs.detach().cpu().numpy())
reconstructs.append(out.detach().cpu().numpy())
```

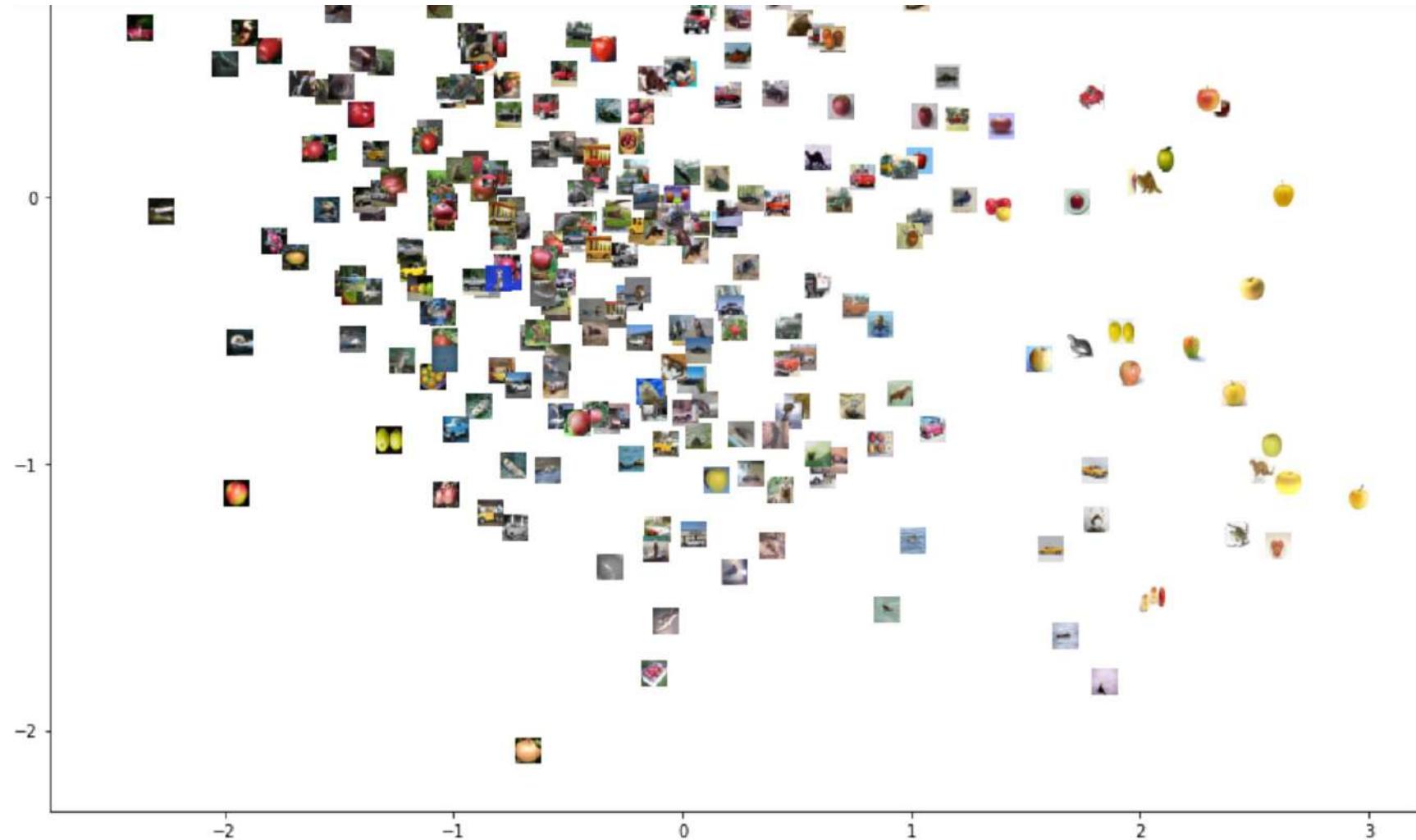
```
projections = PCA(n_components=2).fit_transform(embeddings)

def implot(x, y, image, ax, zoom=1):
    im = OffsetImage(image, zoom=zoom)
    ab = AnnotationBbox(im, (x, y), xycoords='data', frameon=False)
    ax.add_artist(ab)
    ax.update_datalim(np.column_stack([x, y]))
    ax.autoscale()

fig, ax = plt.subplots(1, 1, figsize=(15, 15))
for img, x in zip(images, projections):
    img = img.reshape(32, 32, 3)/255.
    implot(x[0], x[1], img, ax=ax, zoom=0.5)
```

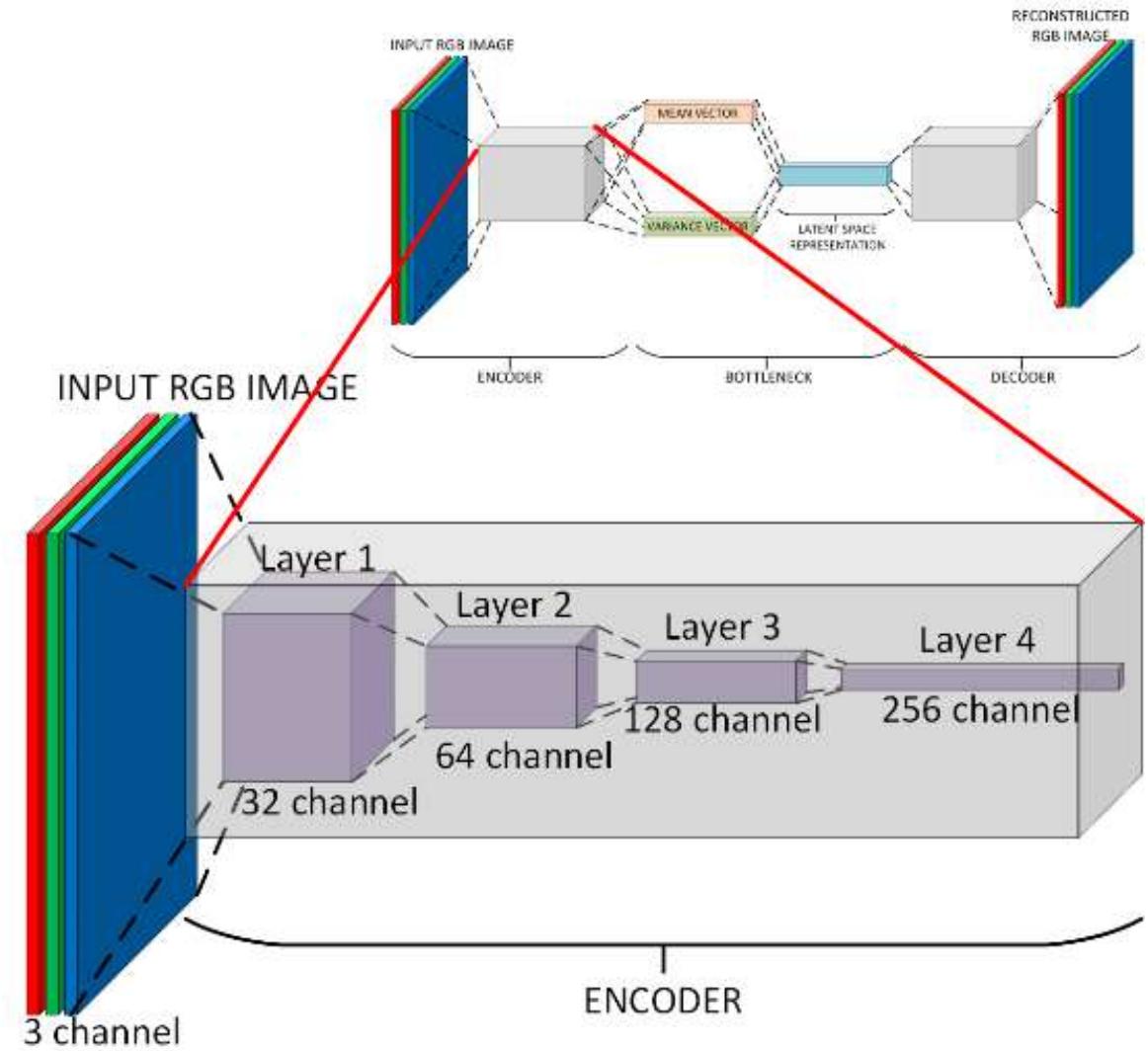
Эмбеддинги

- Отображение каждого примера находятся ближе друг к другу в зависимости от их схожести



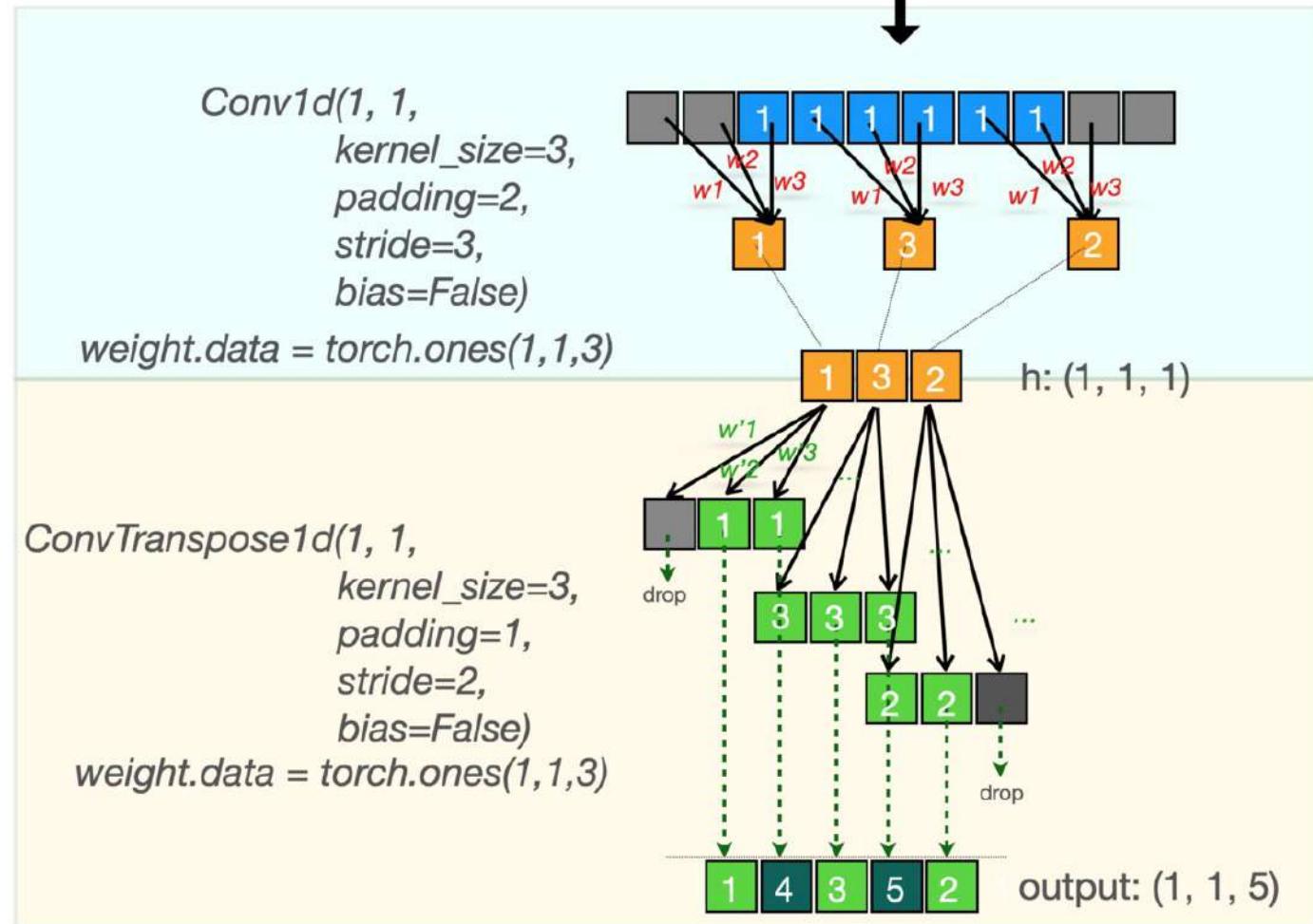
Сверточный кодировщик

- Сверточный кодировщик – это привычная сверточная нейронная сеть
- На самом деле на картинке вариационный автоэнкодер, о нем расскажем позже. Но принцип тот же



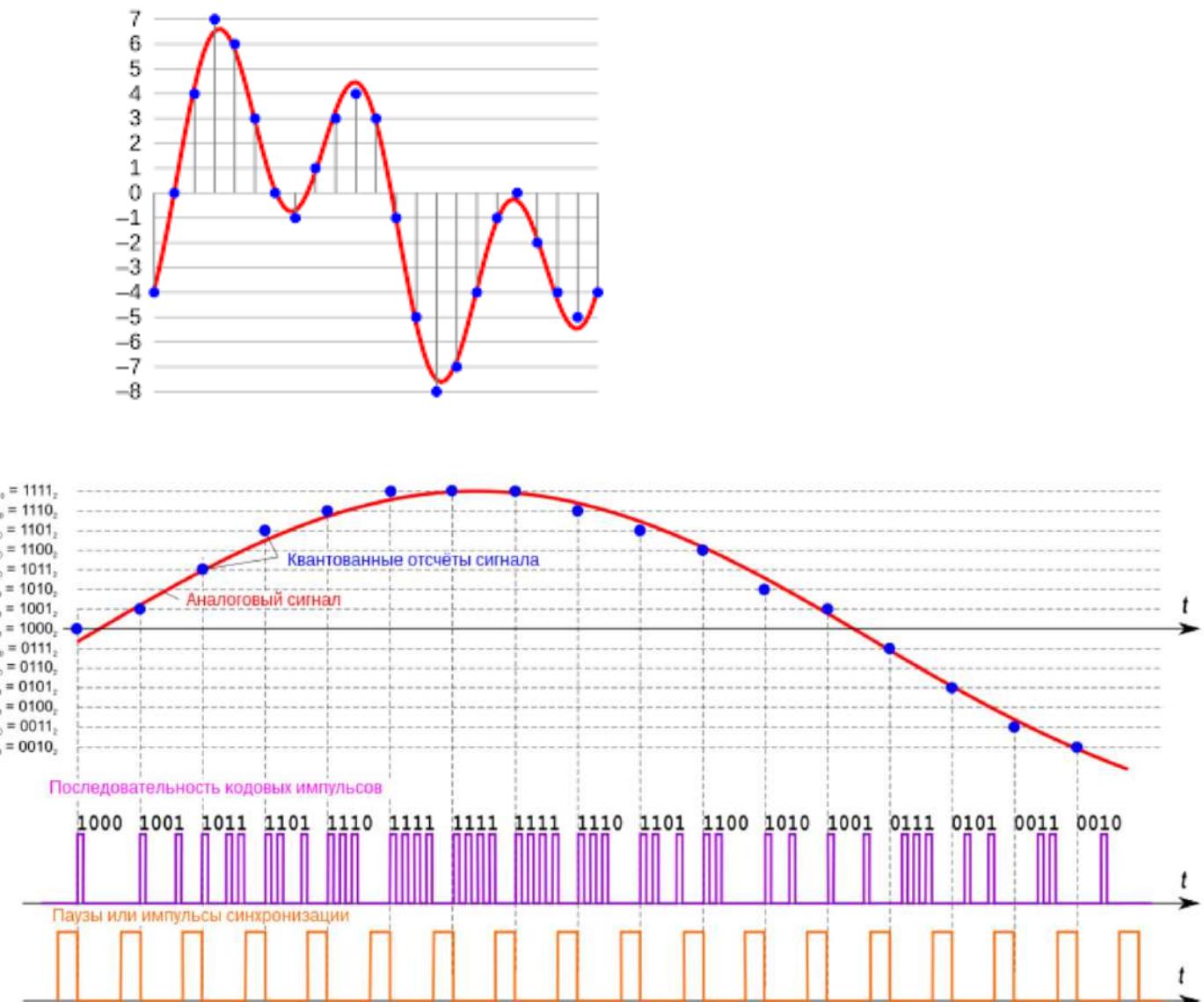
Одномерный автокодировщик

- Сверточный кодировщик – это привычная сверточная нейронная сеть
- Декодировщик – обратные сверточные слои
- Обучаем параметры: их меньше и они тоже разделяемые



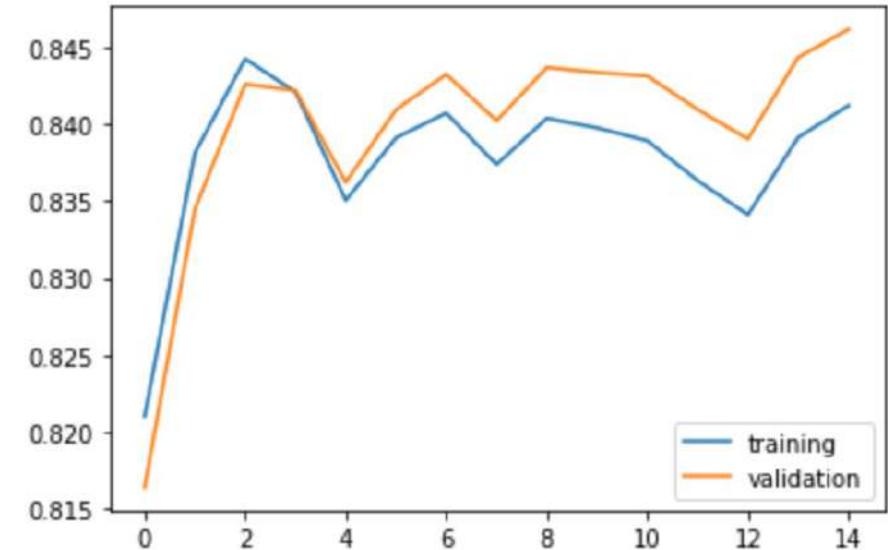
Обработка звука

- wav – формат с использованием импульсно-кодовой модуляции
- В mp3 используется алгоритм сжатия с потерями



Удаление шума из аудио

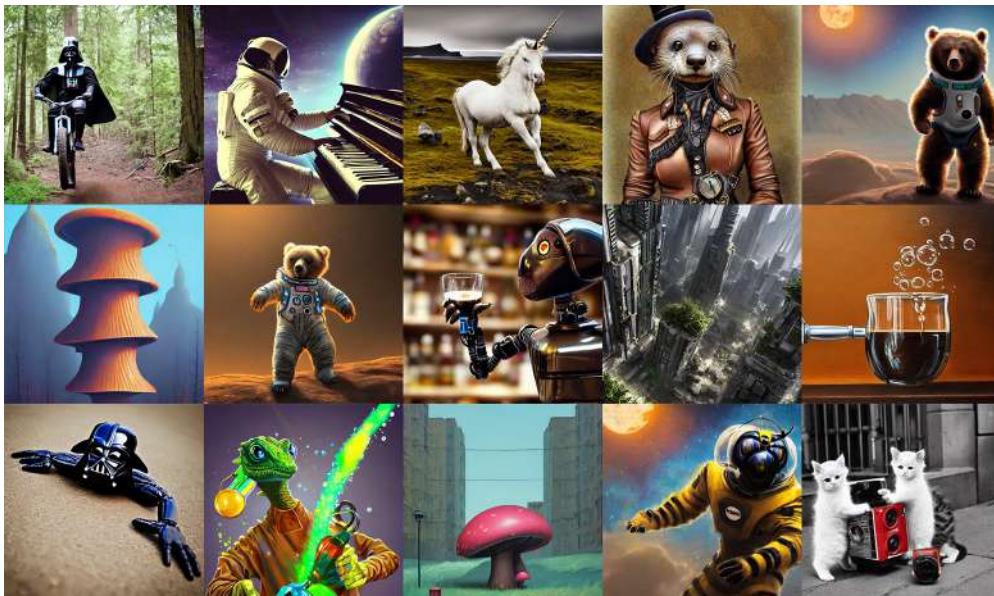
- Используем сверточный автоэнкодер
- В слоях задаем количество каналов и другие настройки сверточных слоев



```
class DenoisingAE(nn.Module):  
    def __init__(self):  
        super().__init__()  
  
        self.encoder = nn.Sequential(  
            nn.Conv1d(2, 256, kernel_size=3, stride=2, padding=1),  
            Mish(),  
            nn.Conv1d(256, 512, kernel_size=3, stride=2, padding=1),  
            Mish(),  
            nn.Conv1d(512, 1024, kernel_size=3, stride=2, padding=1),  
            Mish(),  
        )  
        self.decoder = nn.Sequential(  
            nn.ConvTranspose1d(1024, 512, kernel_size=3, stride=2, padding=1),  
            Mish(),  
            nn.ConvTranspose1d(512, 256, kernel_size=3, stride=2, padding=1),  
            Mish(),  
            nn.ConvTranspose1d(256, 2, kernel_size=3, stride=2, padding=1),  
        )
```

Современные генеративные модели

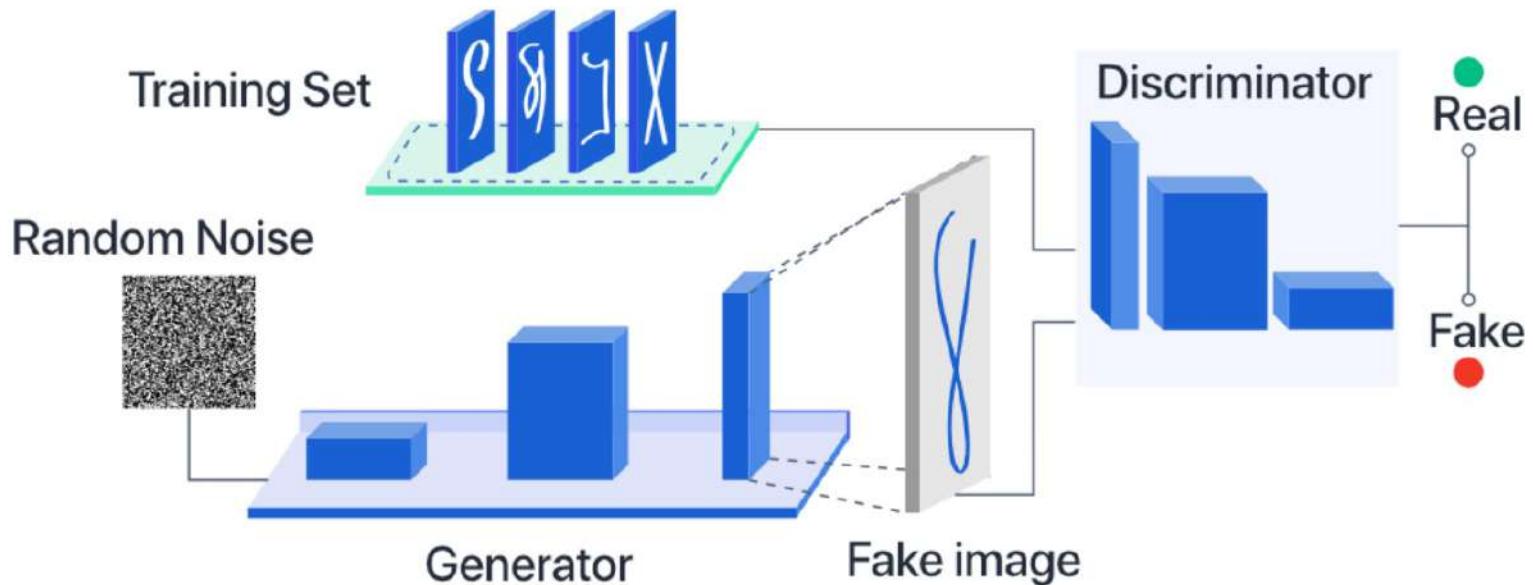
- Dall-E
- Midjourney
- Stable Diffusion



GAN

- Генеративно-состязательная сеть
(Generative adversarial network)

- Дискриминативная отличает правильные от неправильных



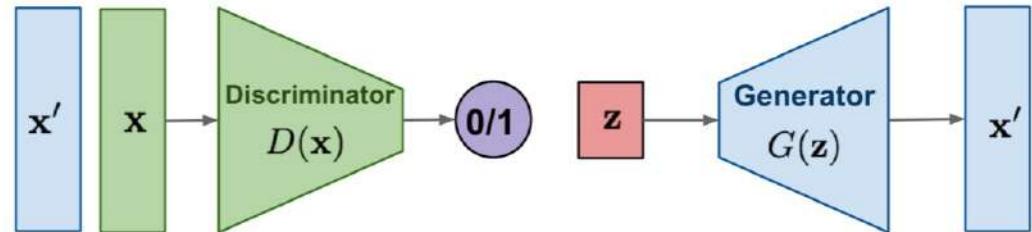
Описал Ян Гудфеллоу в 2014

- Генеративная модель создает образы

Виды генеративных сетей

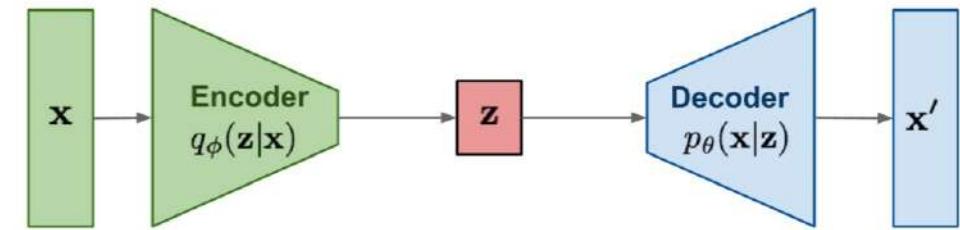
1. Вариационные автоэнкодеры (VAE)

GAN: Adversarial training



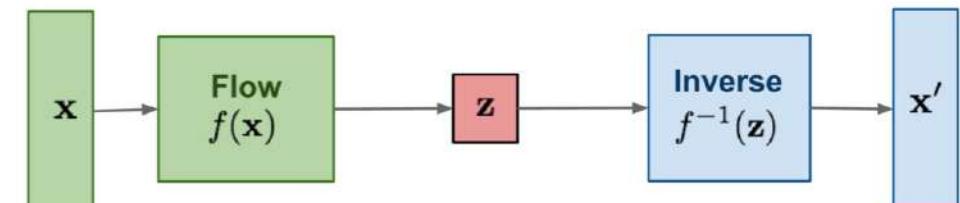
2. Генеративные состязательные сети (GAN).

VAE: maximize variational lower bound



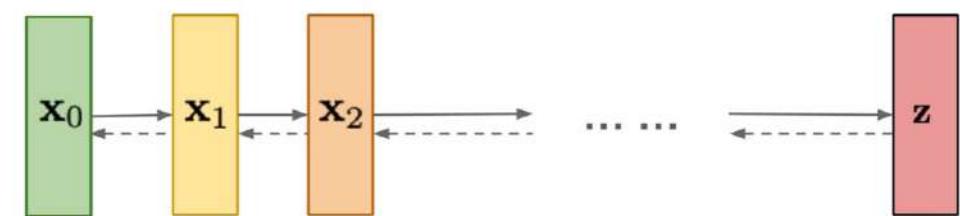
3. Flow-based models

Flow-based models:
Invertible transform of distributions



4. Diffusion (недавняя тенденция)

Diffusion models:
Gradually add Gaussian noise and then reverse

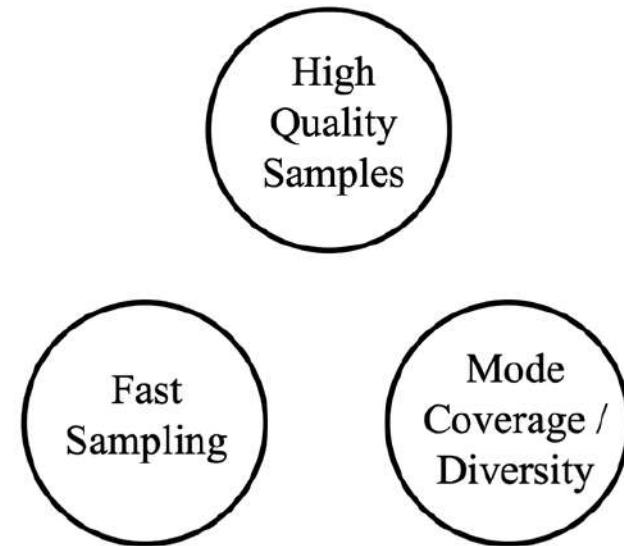


- Источник: <https://lilianweng.github.io/posts/2021-07-11-diffusion-models/>

Какую модель выбрать?

VAE	Поток	GAN	Диффузия
Высокая частота дискретизации.	Высокая частота дискретизации.	Высокая частота дискретизации. Высокое качество генерации образцов.	Высокое качество генерации образцов. Генерация разнообразных образцов
Генерация разнообразных образцов	Генерация разнообразных образцов		

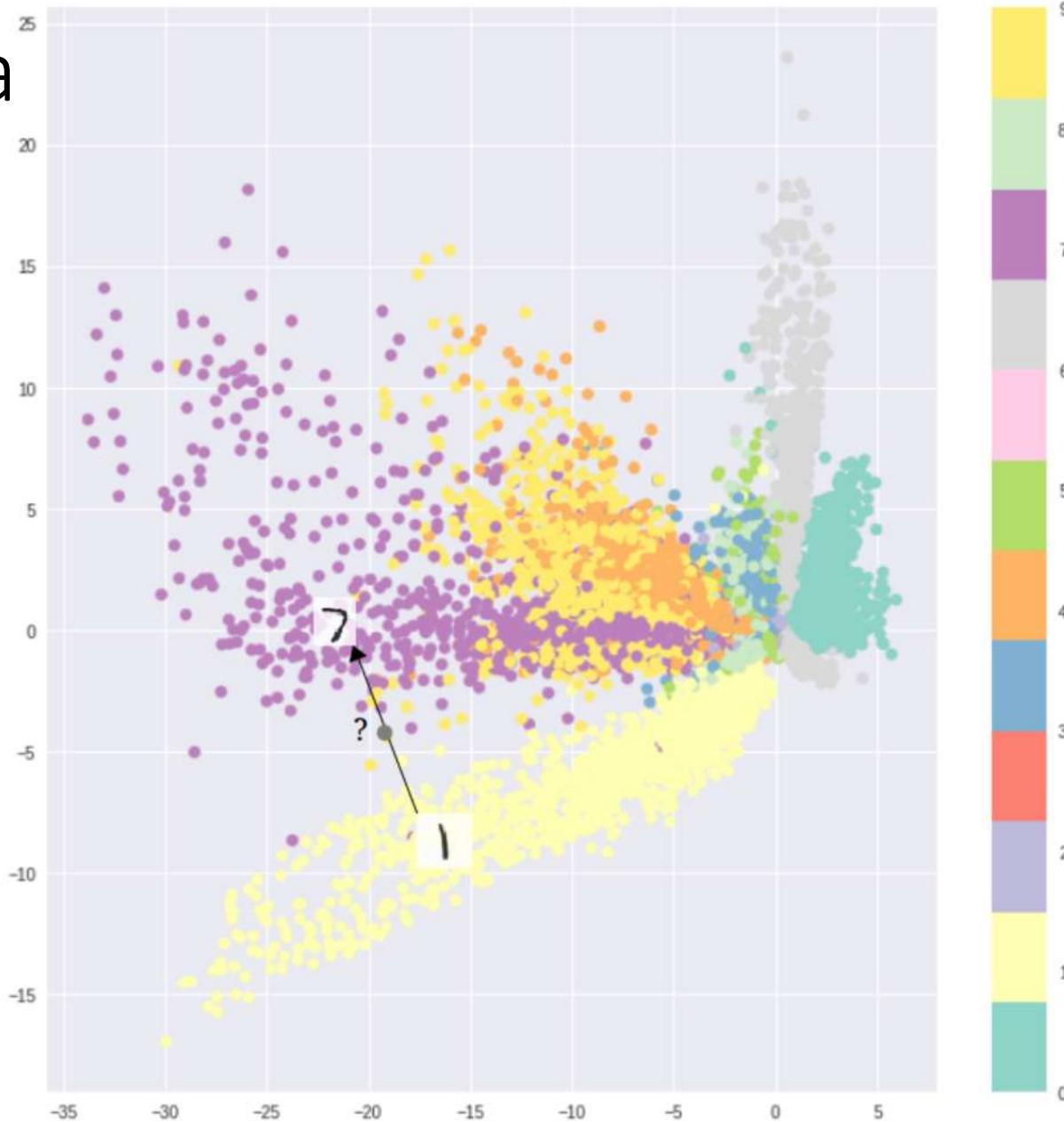
The Generative Learning Trilemma



- Решение трилеммы генеративного обучения
Источник: <https://nvlabs.github.io/denoising-diffusion-gan/>

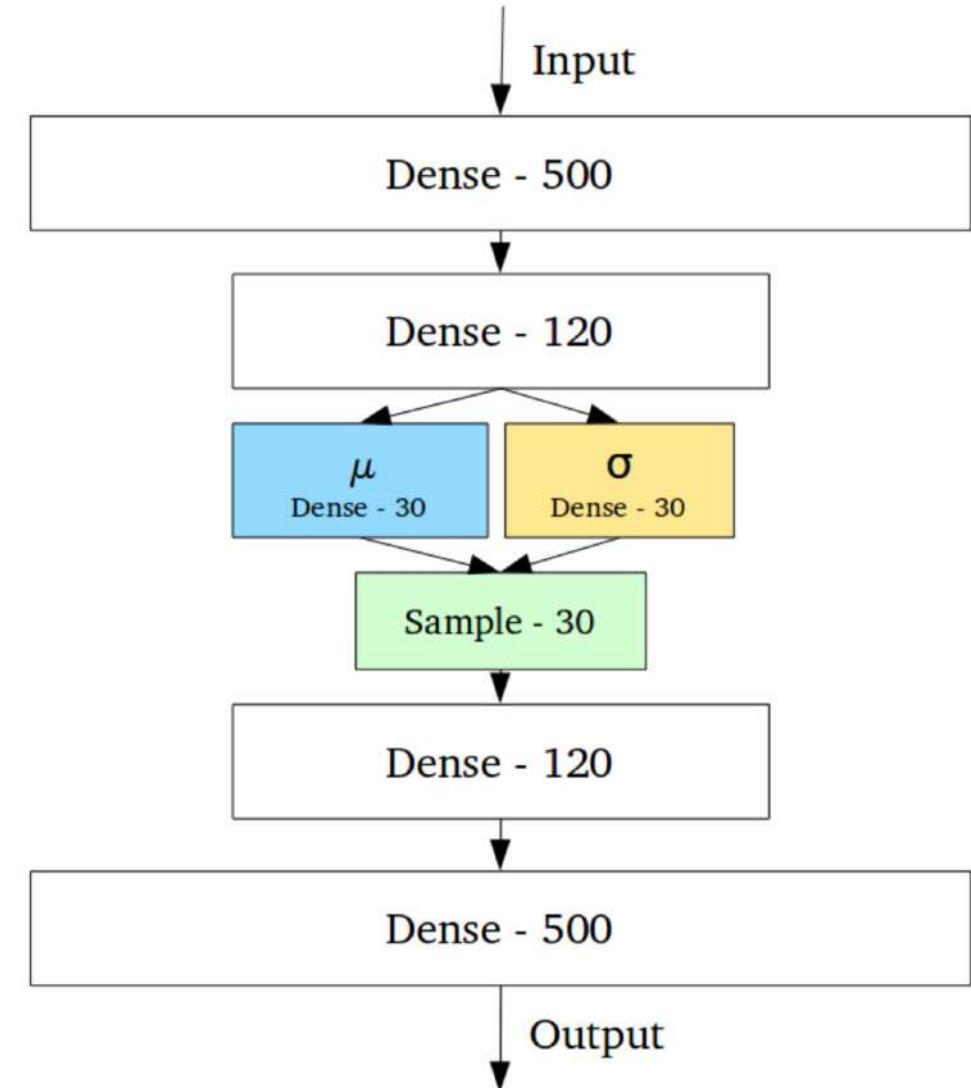
Эмбеддинг автоэнкодера

- Всё замечательно, если вам необходимо просто воспроизвести исходные изображения.
- Однако, если вы строите генеративную модель, ваша цель не простое дублирование изображений.
- Вы хотите получить случайное или видоизмененное изображение, восстановленное из непрерывного скрытого пространства.
- Если пространство имеет разрывы, то декодер выдаст нереалистичное изображение

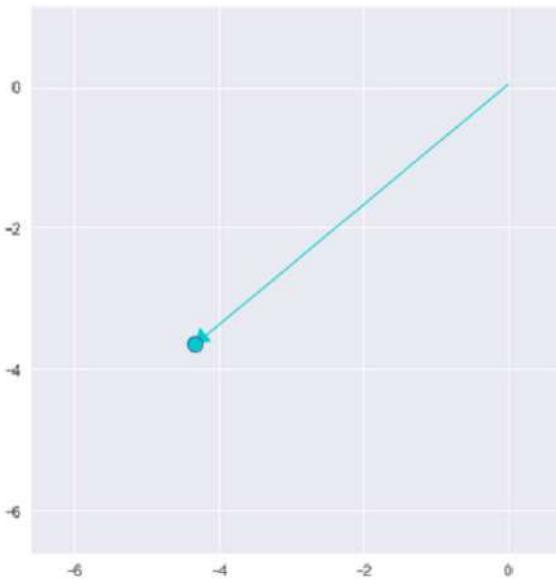


Вариационный автоэнкодер

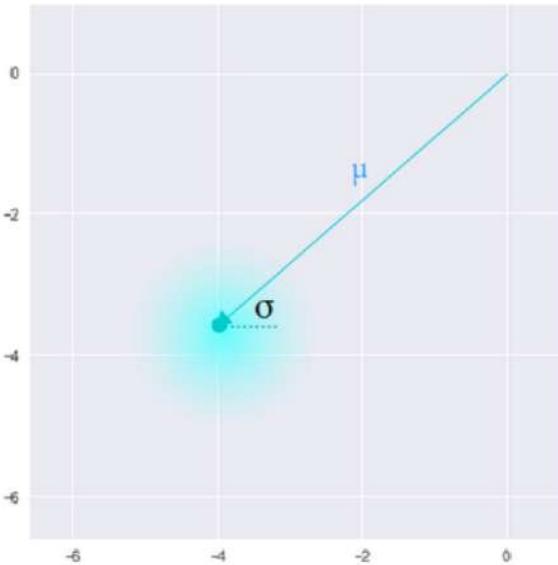
- Вариационный автоэнкодер (Variational Autoencoder – VAE) имеет отличие от стандартного автоэнкодера.
- Их скрытое пространство по построению является непрерывным, позволяя выполнять случайные преобразования и интерполяцию.
- Непрерывность скрытого пространства достигается неожиданным способом: энкодер выдаёт не один вектор размера n , а два вектора размера n – вектор средних значений μ и вектор стандартных отклонений σ .



Эмбеддинг VAE

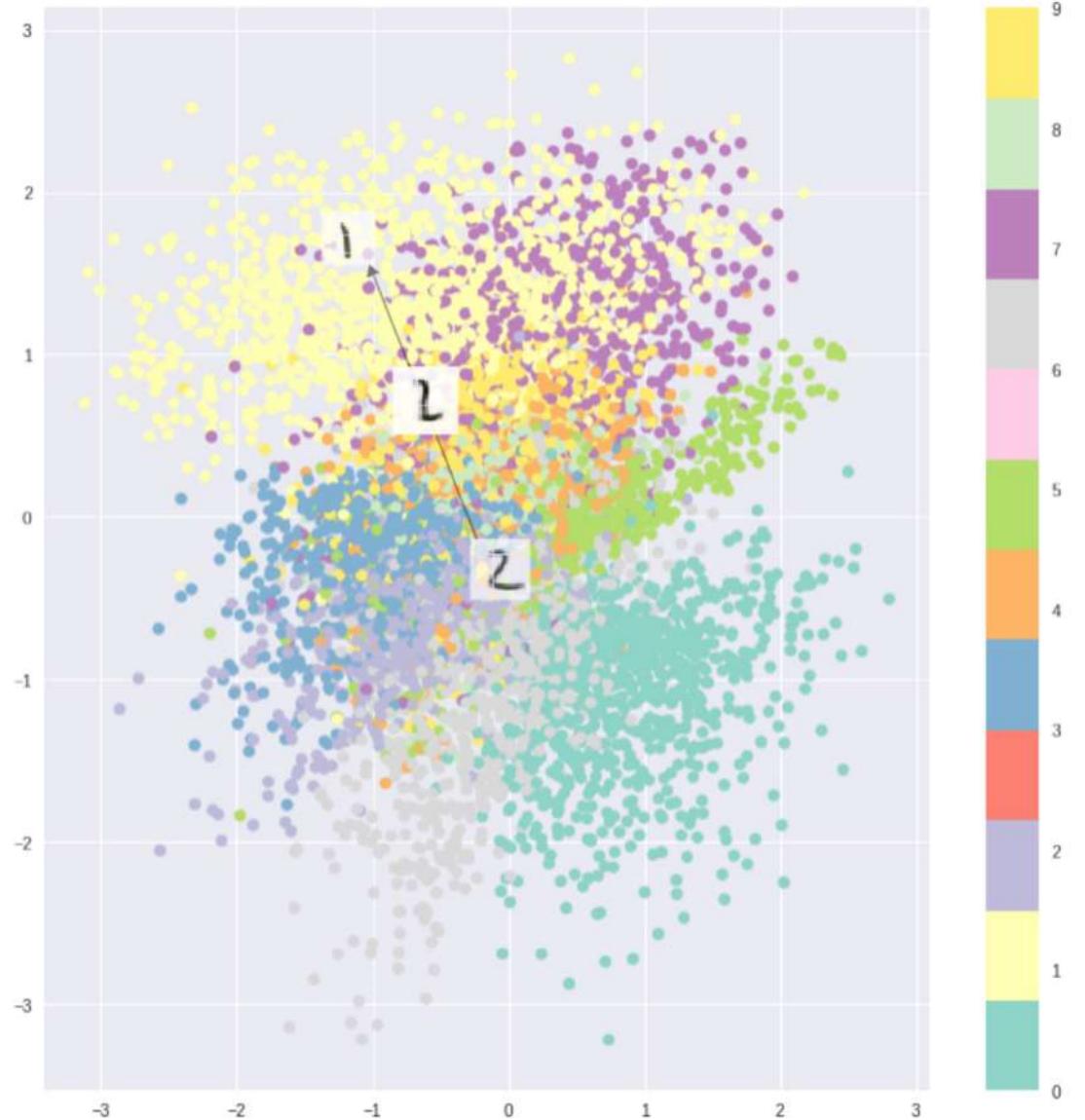


Standard Autoencoder
(direct encoding coordinates)



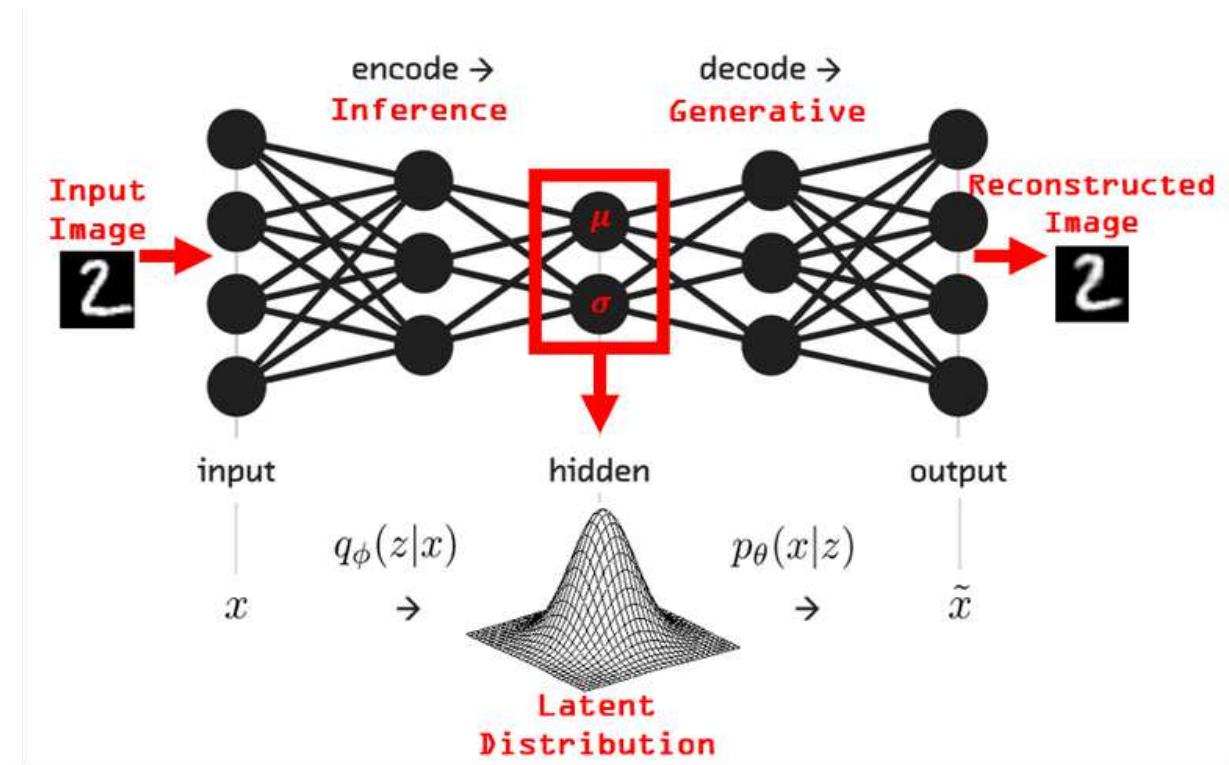
Variational Autoencoder
(μ and σ initialize a probability distribution)

- Если вы хотите восстановить входные данные, вы просто выбираете подходящее распределение и посыпаете его в декодер



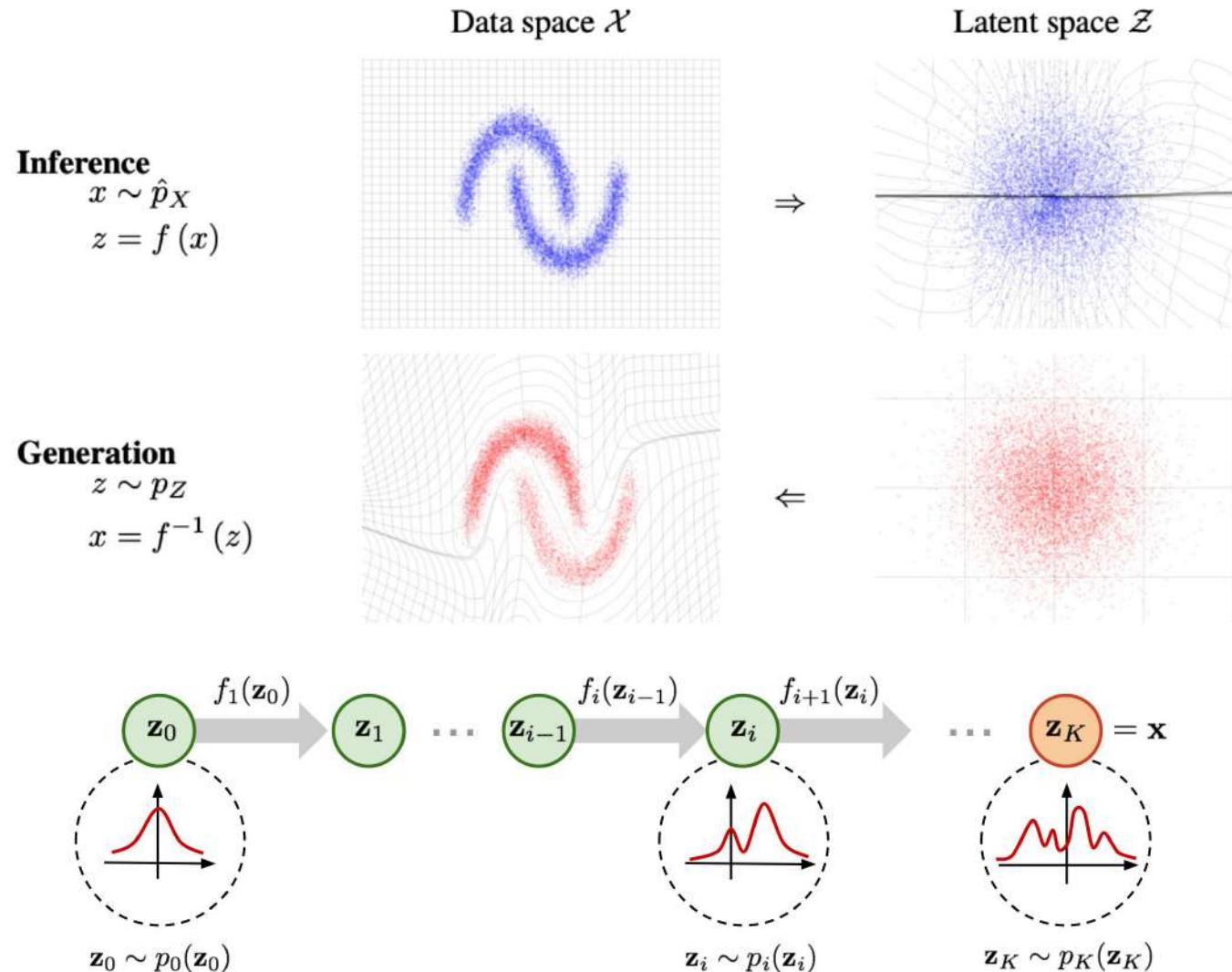
Распределение скрытого состояния

- Исходные изображения имеют свое распределение
- Мы получаем на выходе кодировщика свое распределение и хотим из него получить изображения



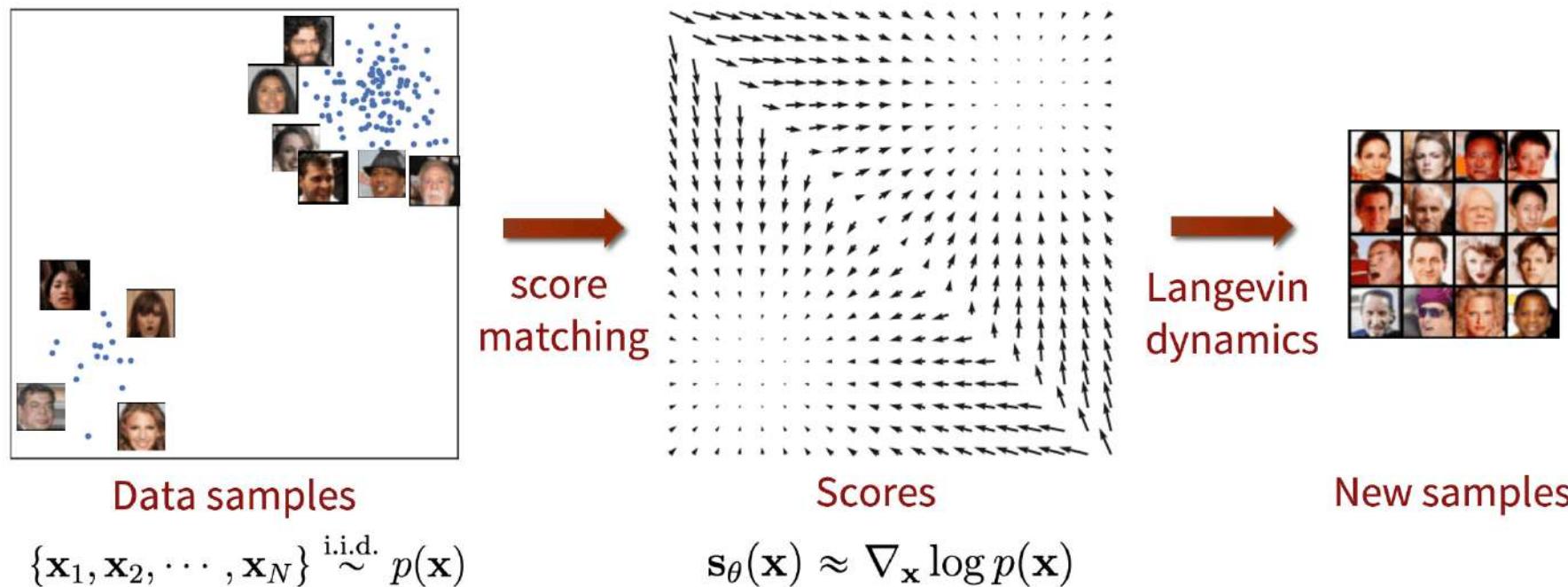
Модели основанные на потоках

- Нам требуется подобрать преобразование, которое позволит отобразить распределение наших изображений в скрытое пространство с Гауссовым распределением
- И мы можем восстановить изображение из скрытого пространства



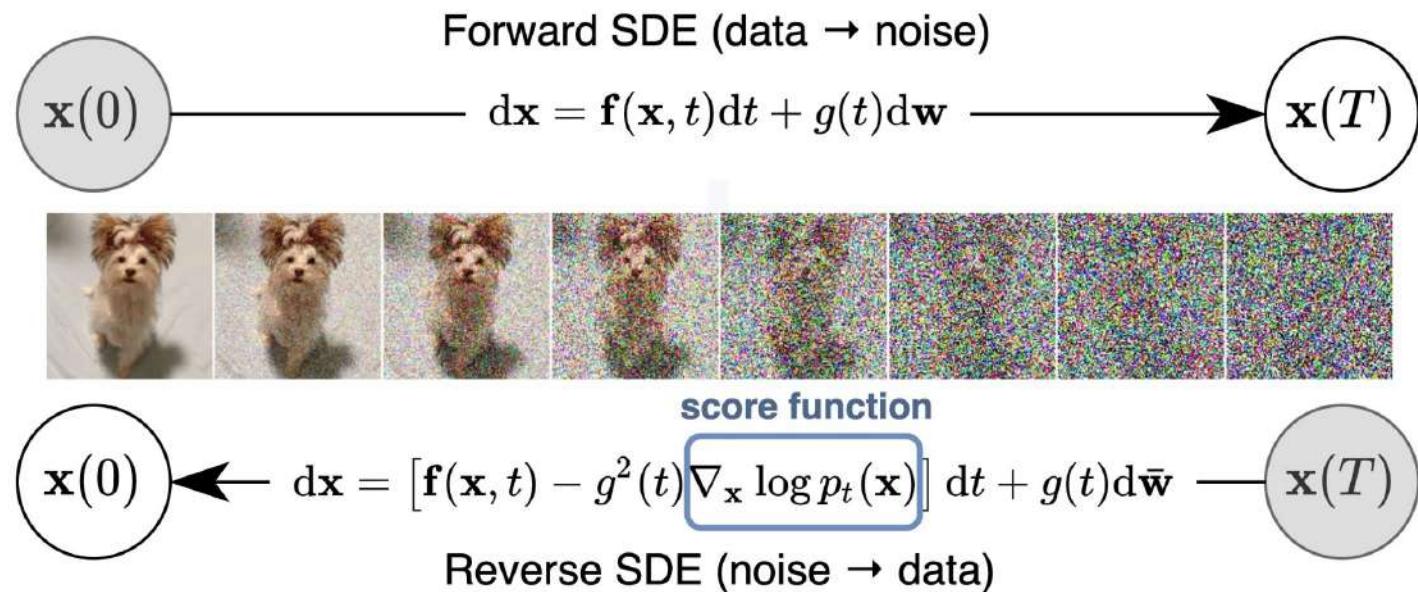
Диффузионные модели

- Появились в 2015, но были хуже GAN. Но в 2021 позволили достичь современных результатов
- Используется Score-функция для обучения



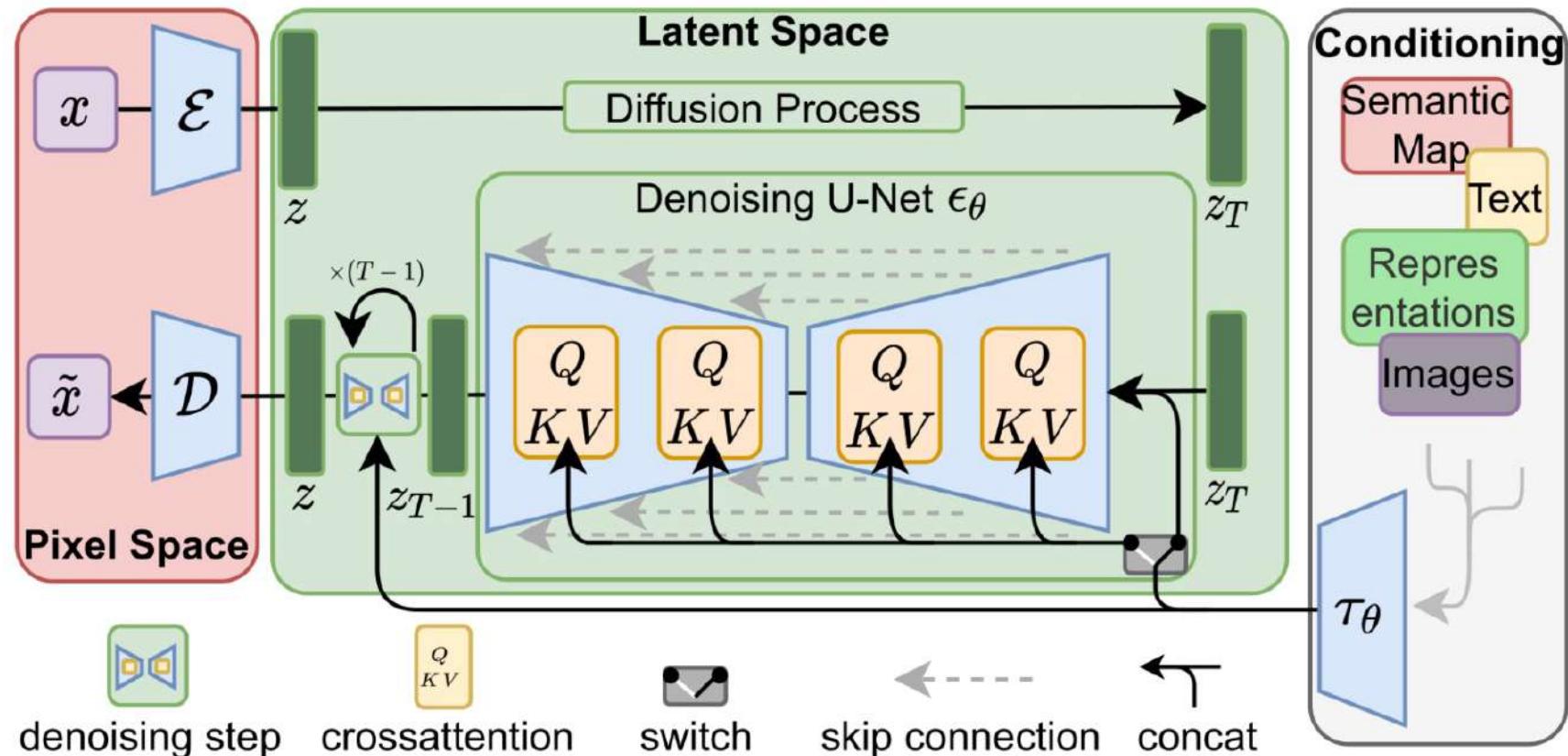
Диффузионные модели

- В диффузионных моделях мы последовательно добавляем Гауссовый шум в исходные данные и учимся восстанавливать его



Stable Diffusion

- Есть кодировщик и декодировщик
- В скрытом пространстве используется диффузионная модель
- Из текста также получаем скрытое состояние и используем в шагах z
- **Модель открытая**

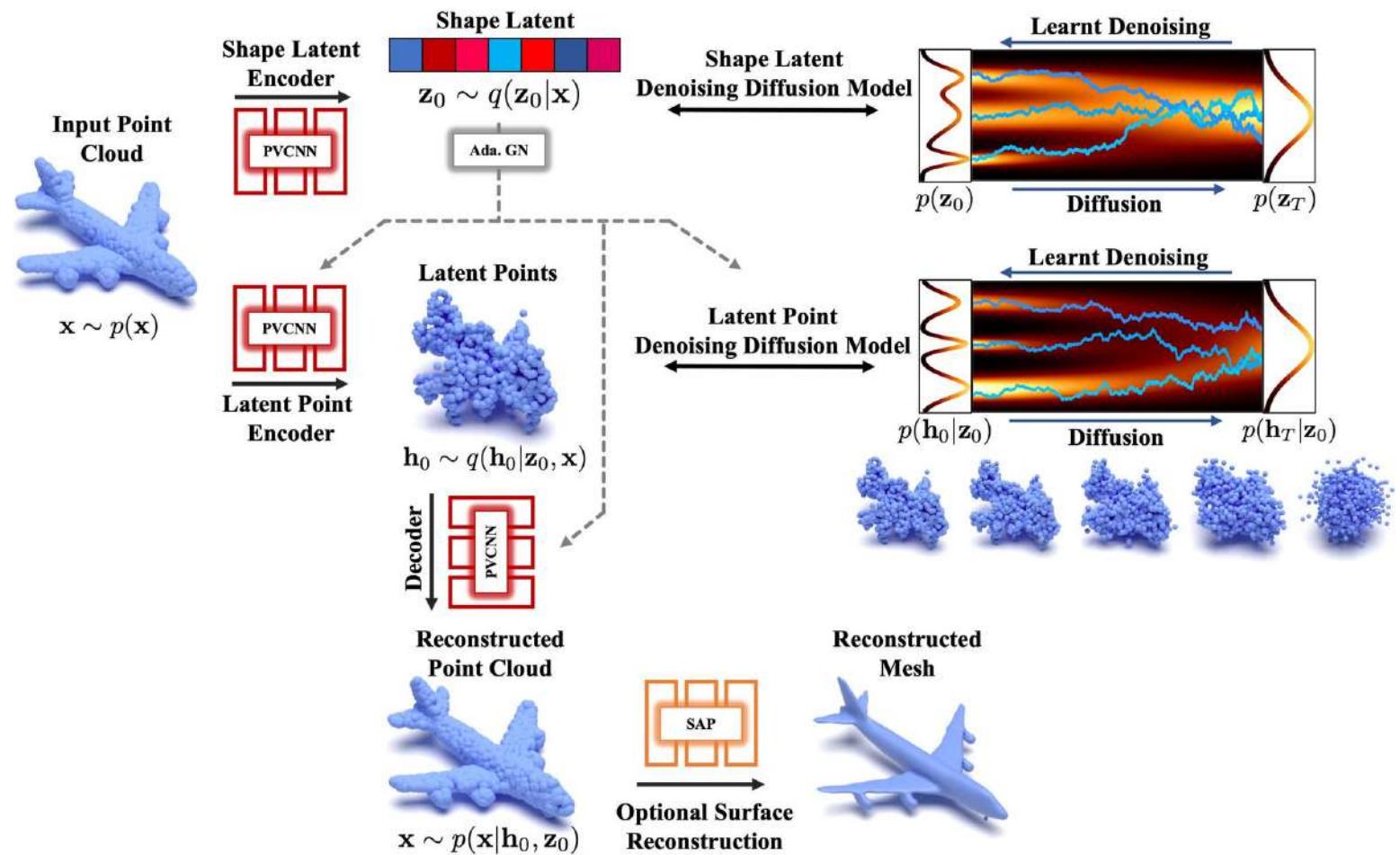


Jay Allamar

<https://jalamar.github.io/illustrated-stable-diffusion/>

Генерация 3D

- Создание новых данных в облаке точек с помощью диффузионных моделей



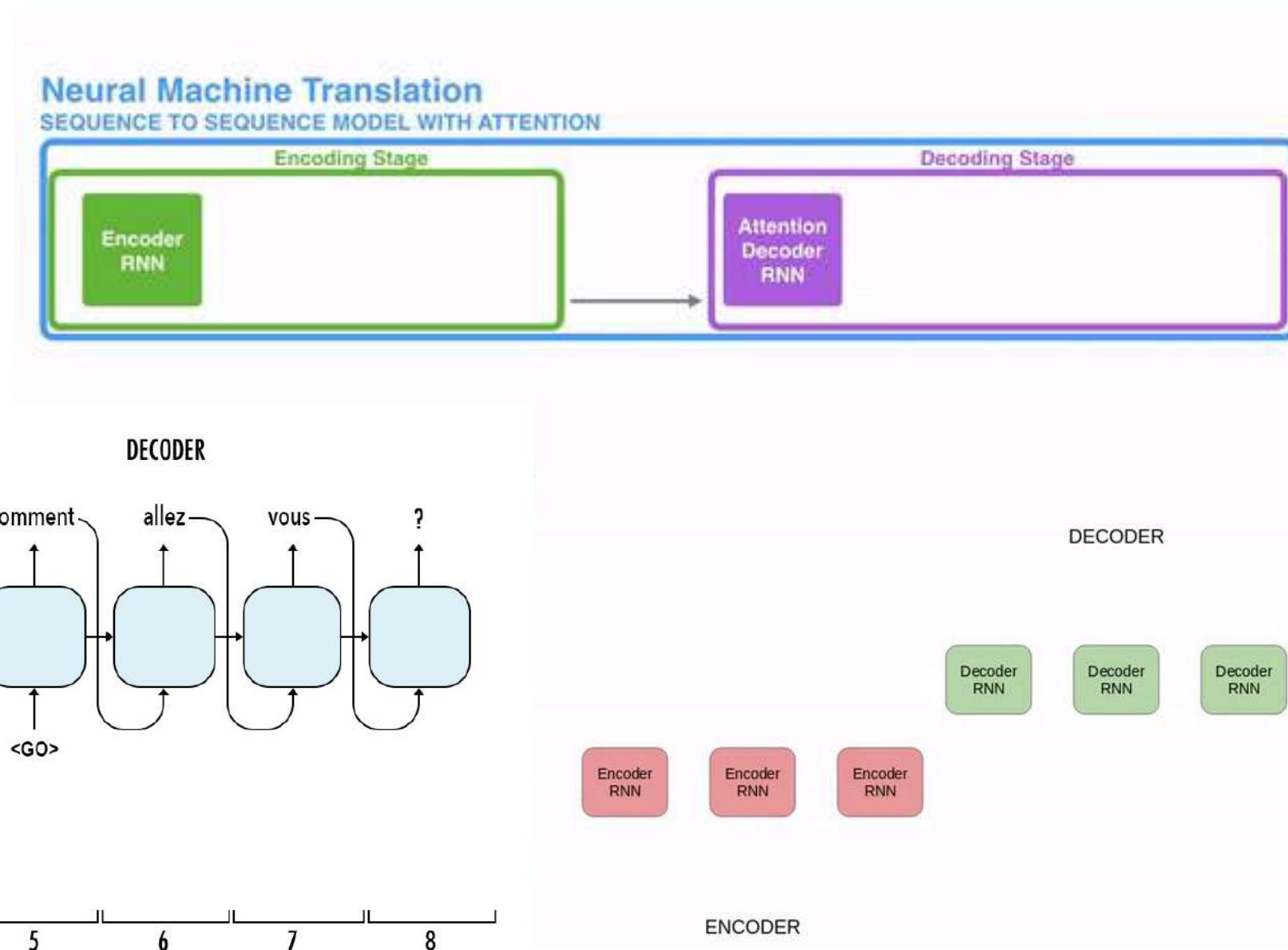
Лекция 8

Трансформеры и механизм внимания

Разработка нейросетевых систем

Seq2seq

- Для машинного перевода
- Состоит из двух частей, например две LSTM
- Можно добавить внешнее внимание



Последовательность -> Последовательность

Машинный перевод

Пример:

ITA: Il gatto si e' seduto sul tappetino.



EN: The cat sat on the mat.

Подход:

seq2seq: имеем один RNN для кодирования исходного предложения, а другой RNN - для предсказания целевого предложения.

В обычной seq2seq для сети decoder мы передаем последнее состояние encoder, но оно не меняется (уже конечное после всех слов) при генерации новых слов с помощью decoder.

Поэтому применим механизм внешнего внимания. Целевой RNN учится (мягко) выравнивать предложение с помощью механизма внимания (обращать внимание на разные слова в зависимости от текста, который уже сгенерировали).

Последовательность -> Последовательность

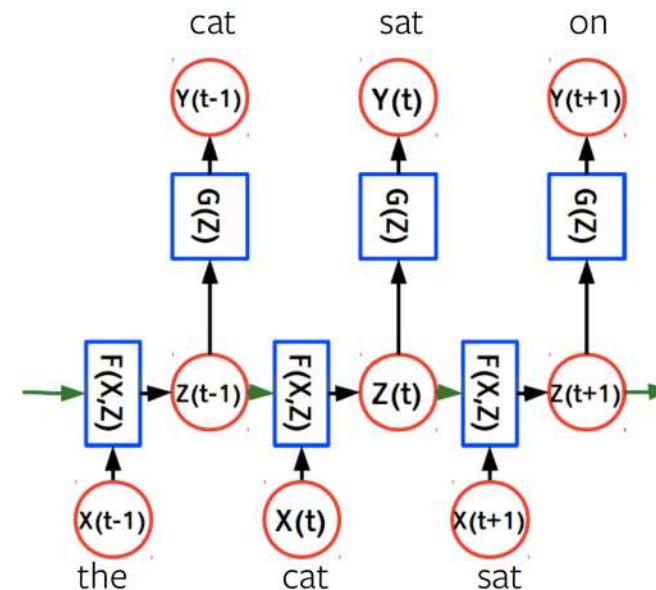
Машинный перевод

Пример:

EN: The cat sat on the mat.

Подход:

Необходимо получить последовательность перевода, генерируя слова одно за другим.
Но остается вопрос – с чего начать перевод?



Y. LeCun's diagram

<https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/>

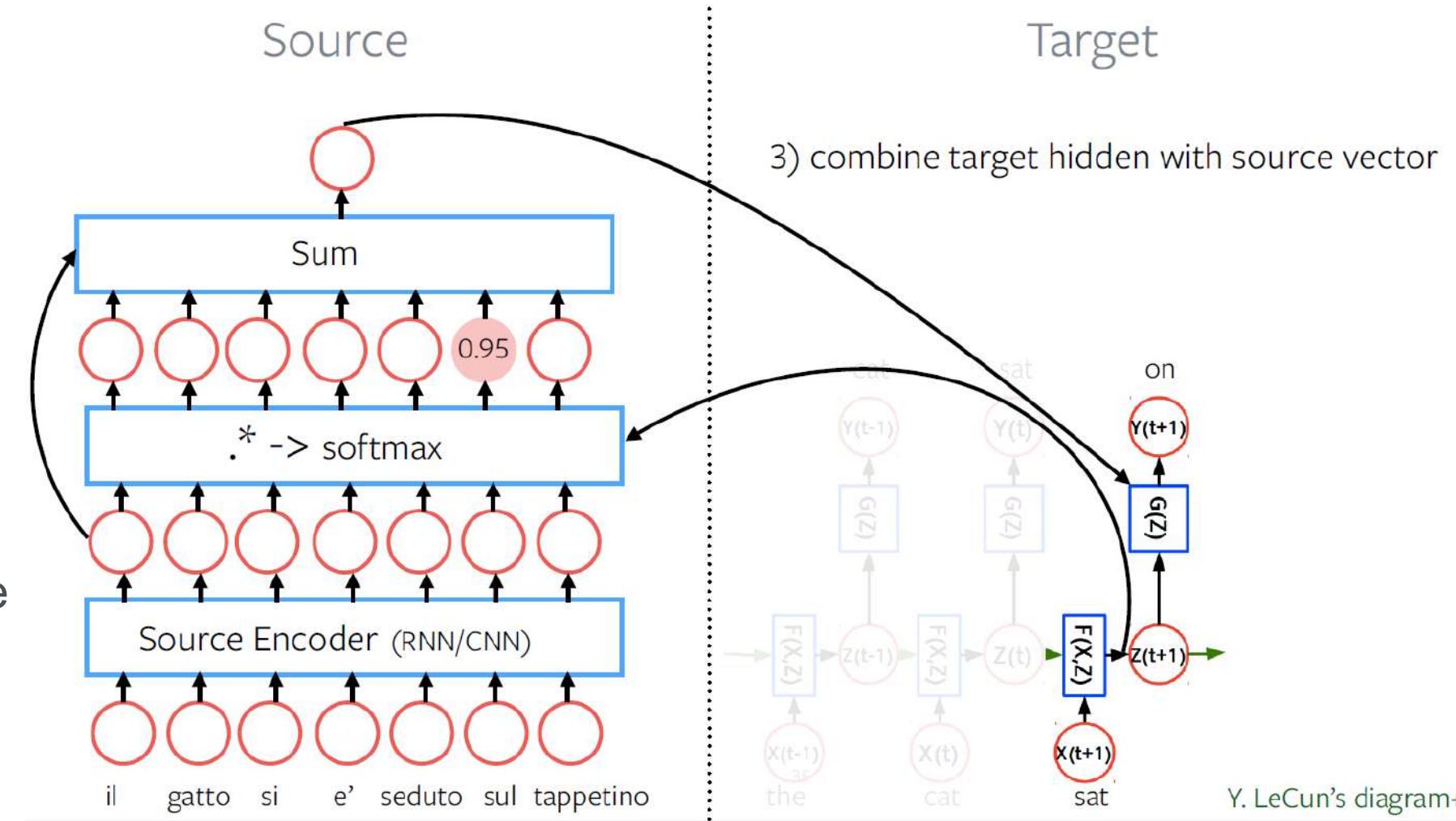
Последовательность -> Последовательность

Внимание RNN

На какое слово из исходного текста нужно обратить внимание?

3 этапа:

- Считаем по всем словам значения в энкодере
- Используем декодер чтобы посчитать внимание
- В генерирующей LSTM используем внимание от всех слов в Encoder

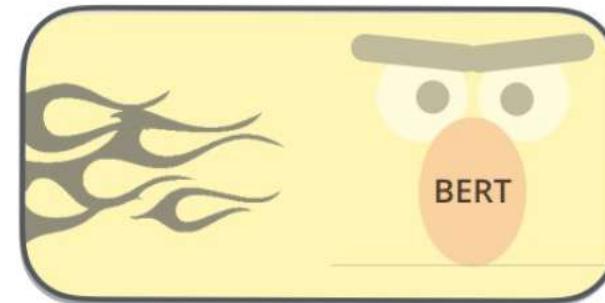


Attention is all you need

- Механизм внимания получил широкое распространение с появлением трансформеров, представленных впервые Google в 2017 году
- Google BERT, Open AI GPT стали применяться повсеместно для решения разных практических задач
- Transformer (encoder+decoder), BERT (только encoder), GPT (только decoder)



<https://jalammar.github.io/illustrated-transformer/>



<https://jalammar.github.io/illustrated-bert/>

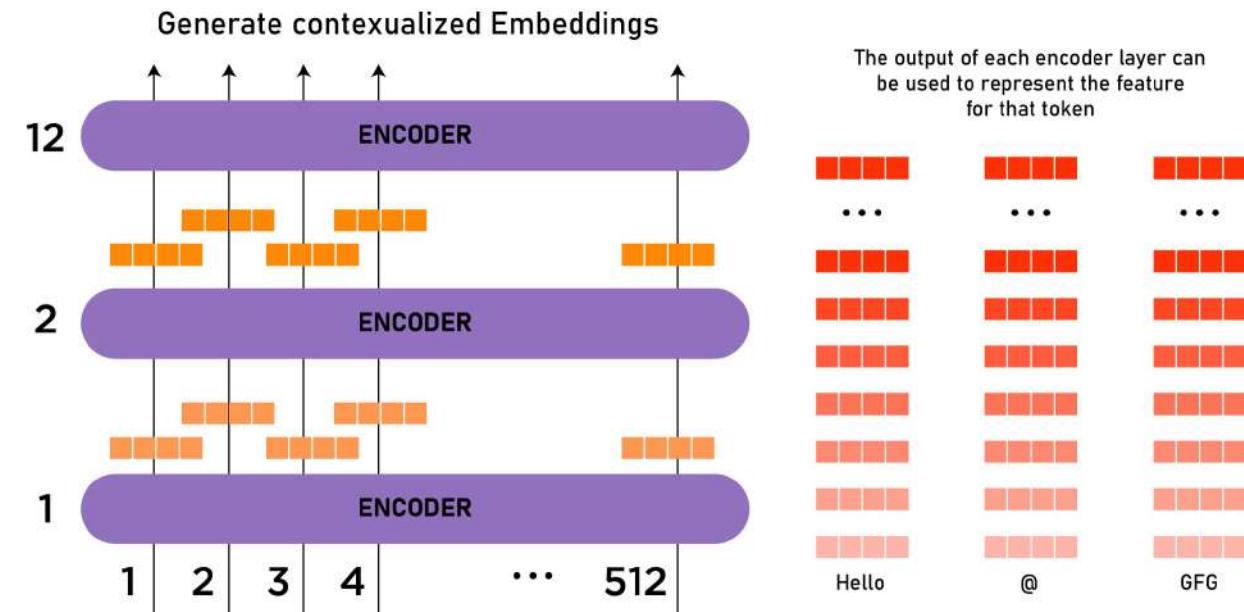
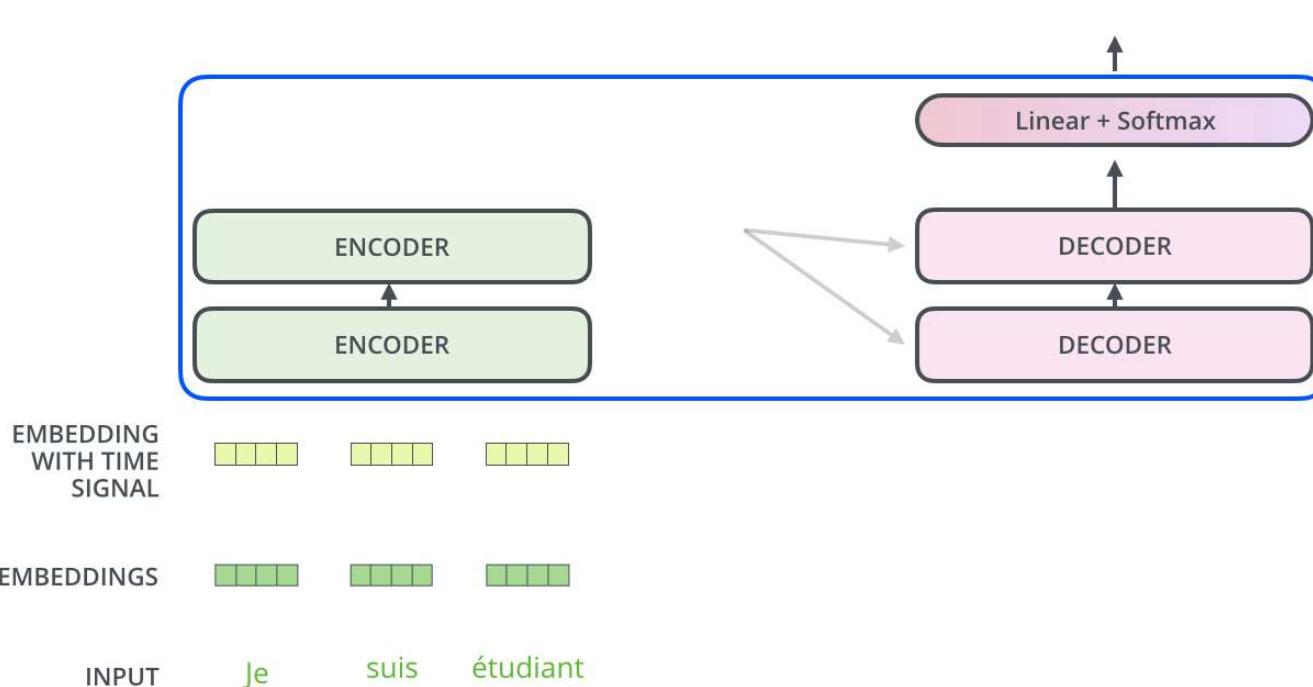
<https://arxiv.org/abs/1706.03762>

Трансформер

- Трансформер с механизмом внимания
- BERT, GPT

Decoding time step: 1 2 3 4 5 6

OUTPUT



BERT

- Если раньше в Word2vec мы создавали один и тот же вектор для одного слова
- BERT теперь дает нам эмбеддинг для текста, мы можем кодировать текст в вектор и сравнивать их (RAG)

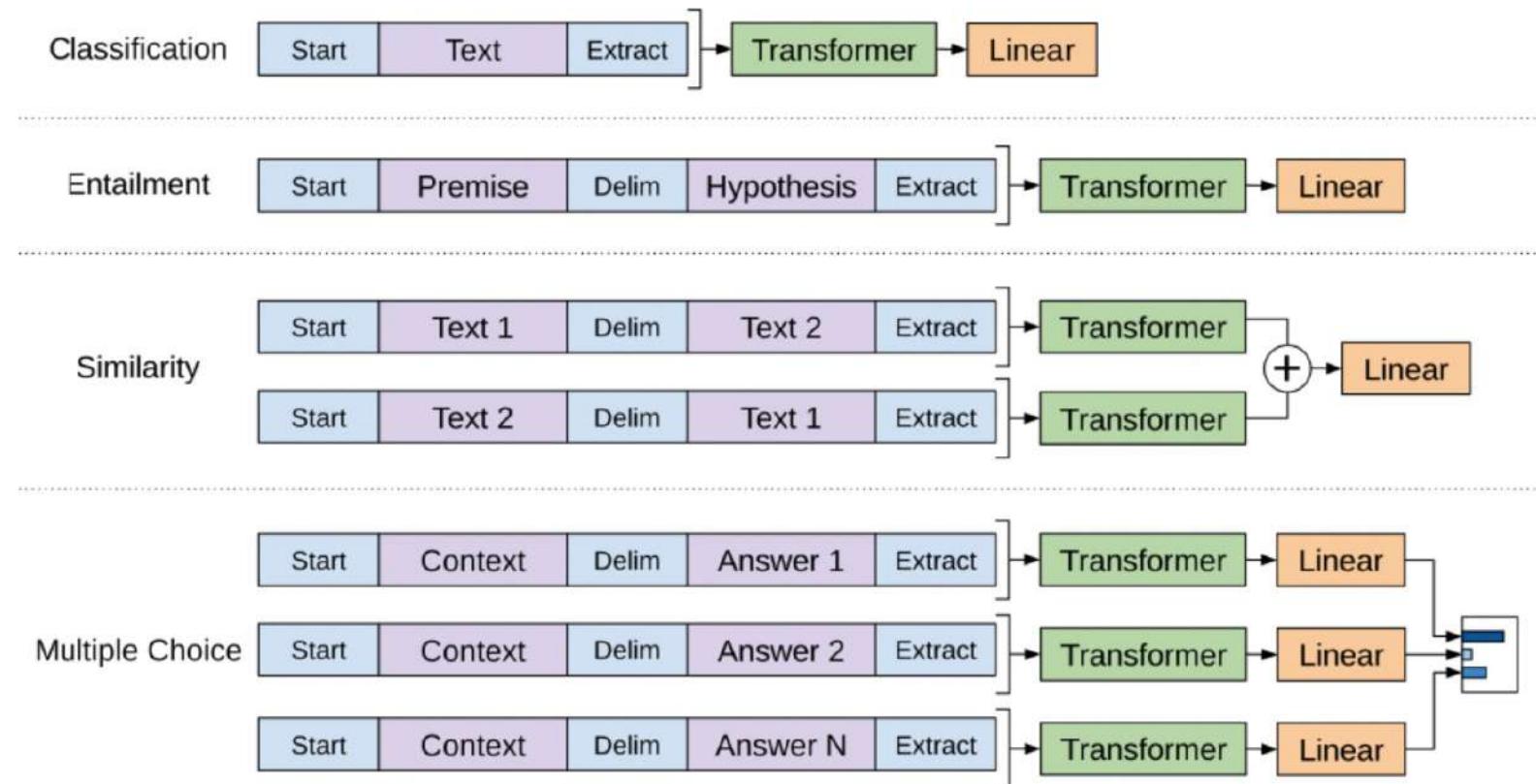
Применение трансформеров

- Статья от OpenAI указывает на ряд трансформаций, делающих возможным обработку входов при решении разнообразных задач

- Пример использования BERT: мы получаем вектор из текста

- Теперь тексты можно сравнить, классифицировать и тд

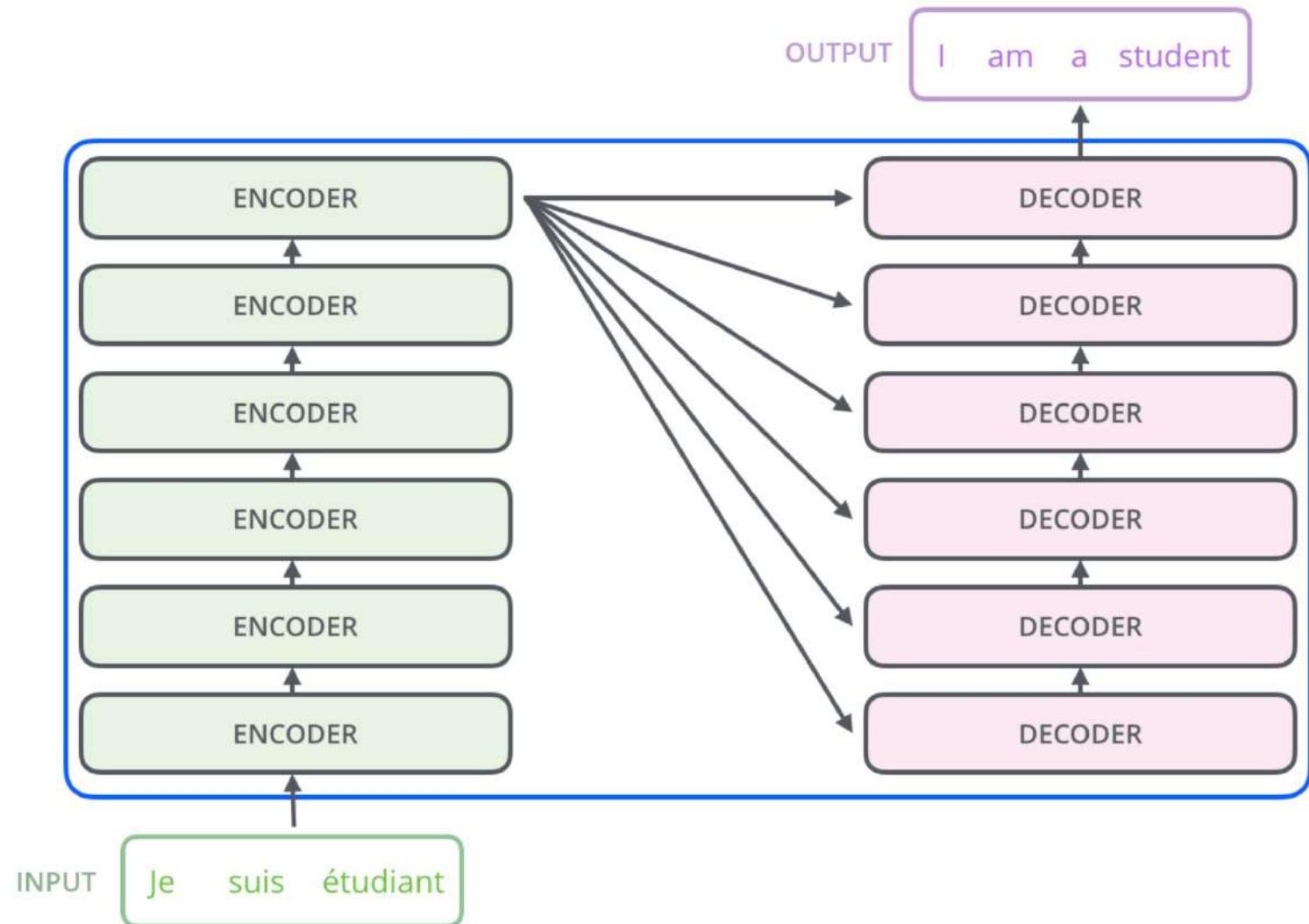
- Мы можем обучить свою очень простую модель на выходе – transfer learning



Архитектура Transformer

Рассмотрим знакомую задачу:

- У нас есть последовательность слов
- Мы должны сгенерировать новую последовательность слов (на другом языке)



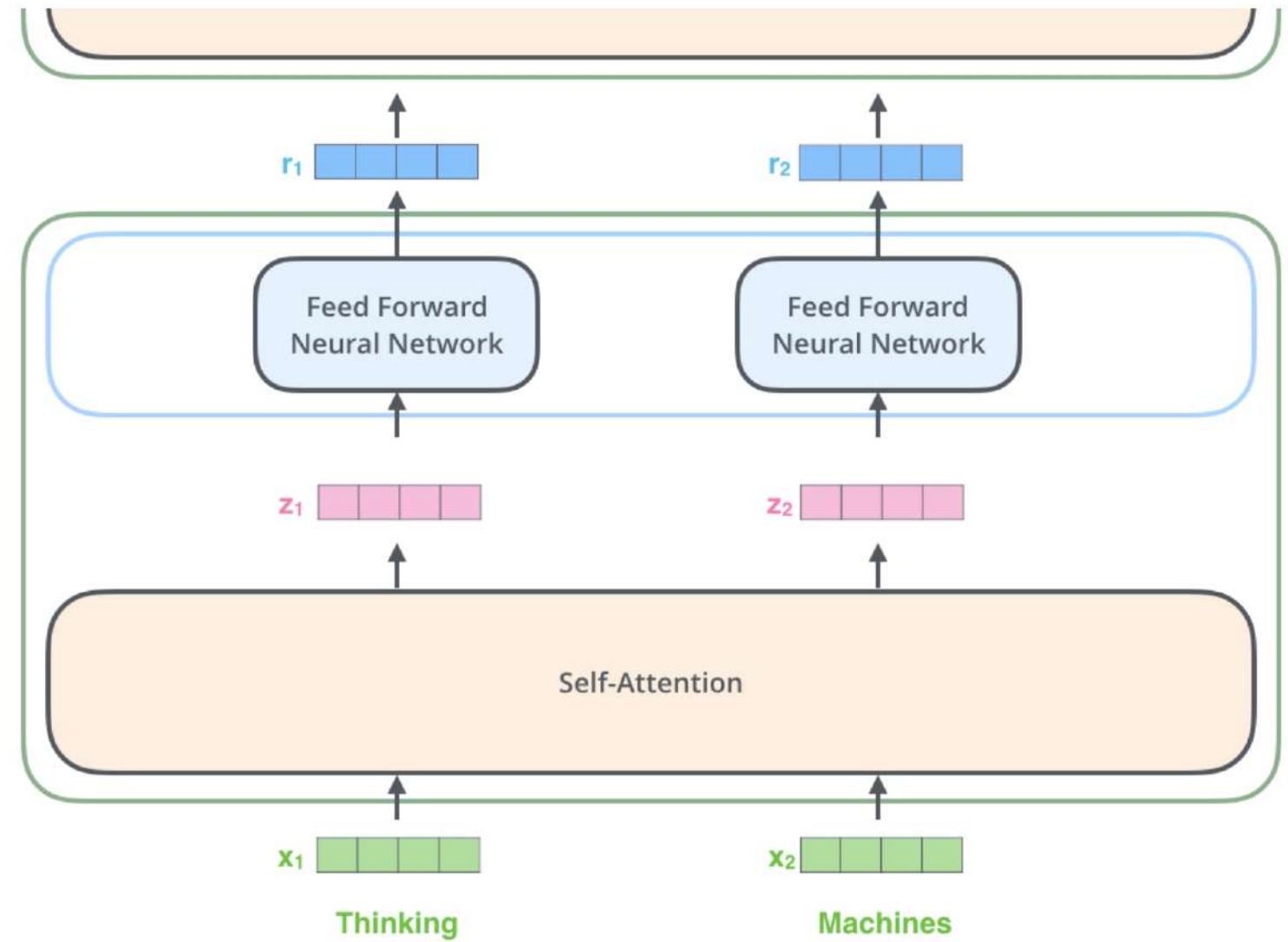
Кодировщик

С помощью
специальность слоя
внутреннего
внимания мы из
эмбеддинга слова
будем получать
новый вектор z

Так повторяем
несколько раз

ENCODER #2

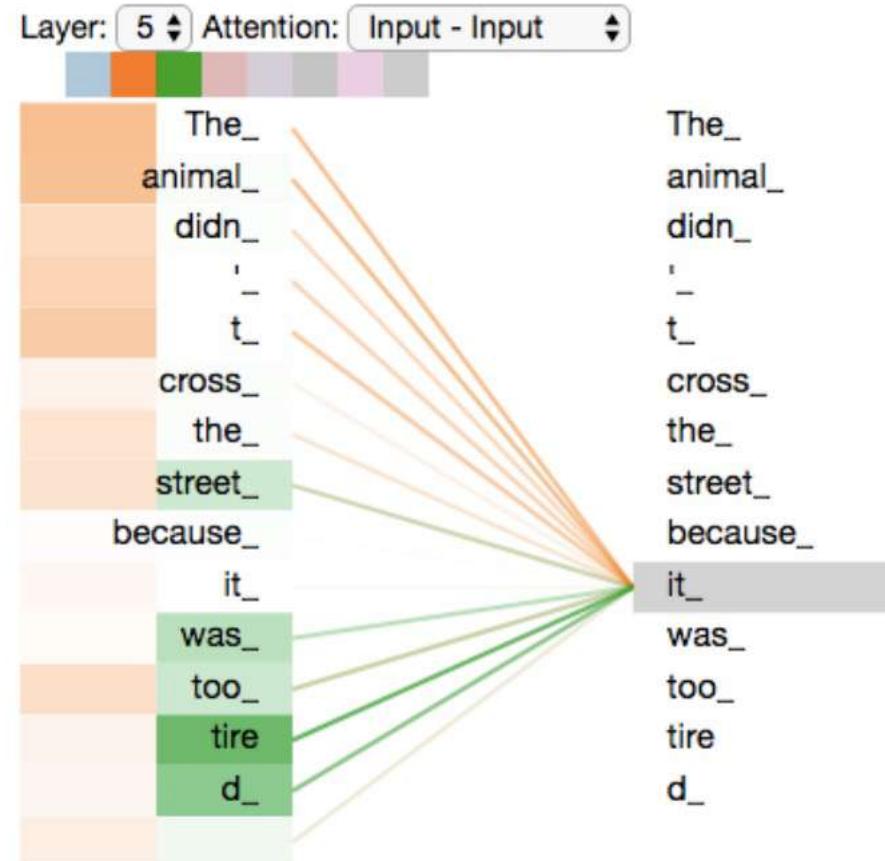
ENCODER #1



Внутреннее внимание

Когда мы кодируем слово «it», одна голова внимания больше всего фокусируется на «животном», в то время как другая фокусируется на «усталом»

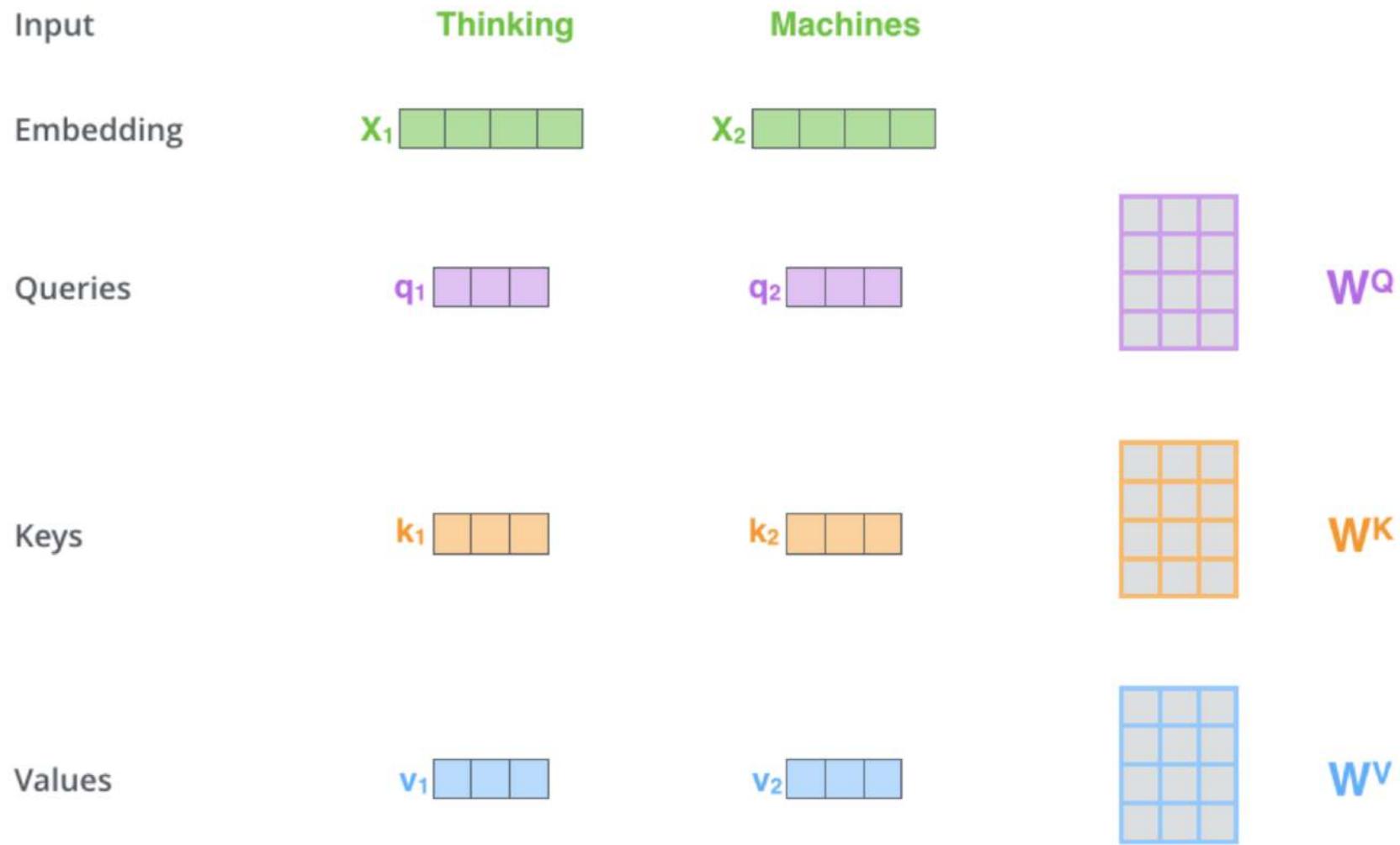
Представление модели слова «it» сочетает в себе часть представления как «животного», так и «усталого»



Внутреннее внимание

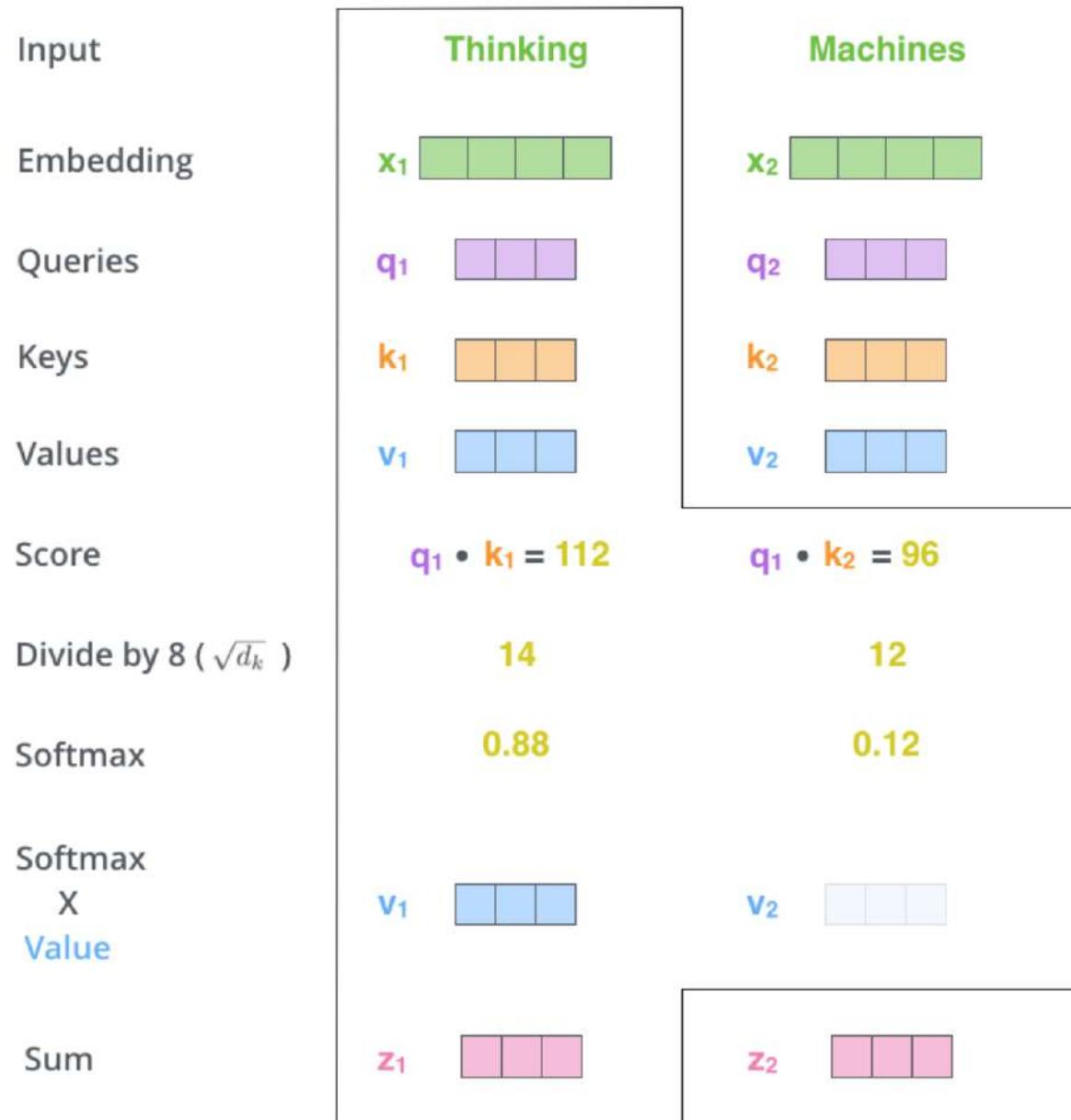
Перемножая x_1 на матрицу весов WQ получаем q_1 , вектор "query", связанный с этим словом.

В итоге мы создаем проекцию "query", "key" и "value" каждого слова во входном предложении.



Query, Key, Value

- Каждая строка в матрице X соответствует слову во входном предложении
- Мы определяем какой в итоге результат будет для данного слова: это его вклад и вклад других слов в предложении



Несколько слов

Каждая строка в матрице X соответствует слову во входном предложении

А количество строк – это размер эмбеддинга для каждого слова

Запишем в матричной форме:

$$\text{softmax} \left(\frac{\begin{matrix} \text{Q} & \text{K}^T \\ \times & \end{matrix}}{\sqrt{d_k}} \right) \text{V}$$

= Z

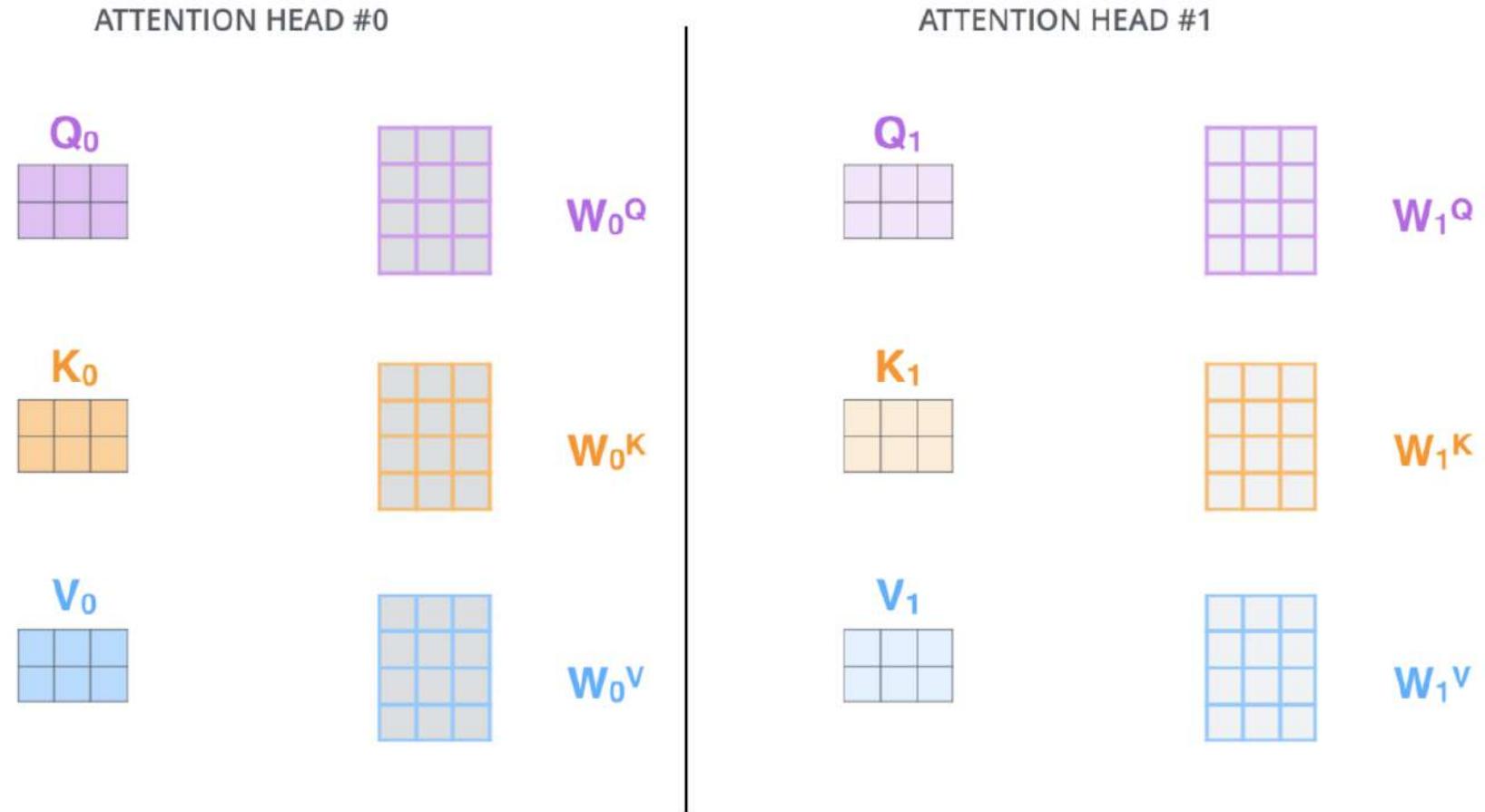
$$X \times W^Q = Q$$

$$X \times W^K = K$$

$$X \times W^V = V$$

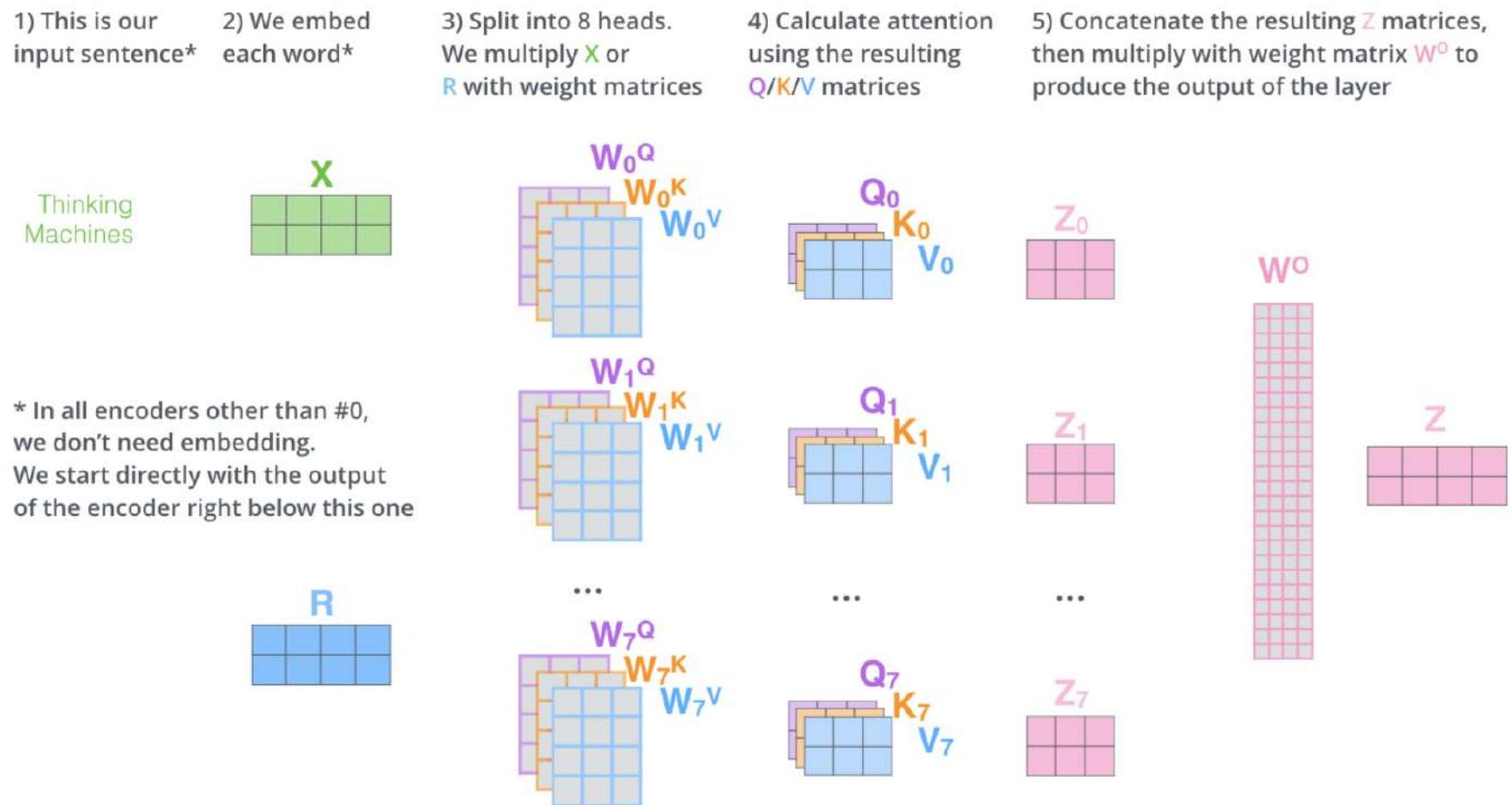
Attention heads

- При многоголовом внимании мы обрабатываем отдельные матрицы весов Q/K/V для каждой головы

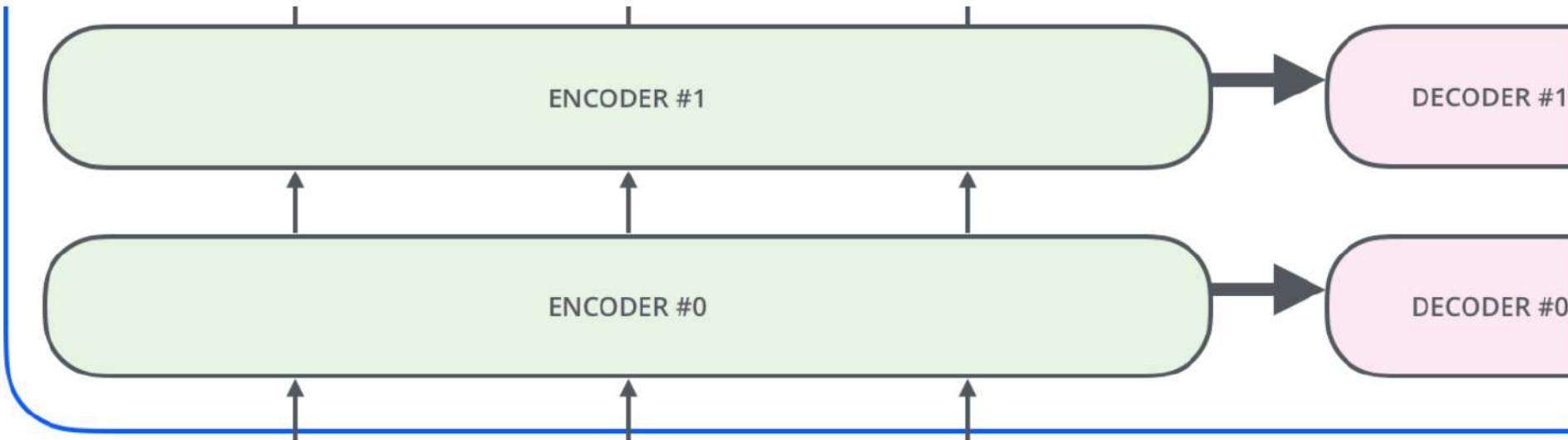


Attention result

- Когда мы получили результат для каждой головы нам нужно его объединить
- Используем еще одну матрицу W (которую тоже обучаем)
- Это пример просто линейной регрессии



Позиции



EMBEDDING
WITH TIME
SIGNAL

$$x_1 \begin{array}{|c|c|c|c|} \hline \text{yellow} & \text{yellow} & \text{yellow} & \text{yellow} \\ \hline \end{array}$$

=

$$t_1 \begin{array}{|c|c|c|c|} \hline \text{orange} & \text{orange} & \text{orange} & \text{orange} \\ \hline \end{array}$$

+

POSITIONAL
ENCODING

$$x_1 \begin{array}{|c|c|c|c|} \hline \text{green} & \text{green} & \text{green} & \text{green} \\ \hline \end{array}$$

$$x_2 \begin{array}{|c|c|c|c|} \hline \text{yellow} & \text{yellow} & \text{yellow} & \text{yellow} \\ \hline \end{array}$$

=

$$t_2 \begin{array}{|c|c|c|c|} \hline \text{orange} & \text{orange} & \text{orange} & \text{orange} \\ \hline \end{array}$$

+

$$x_2 \begin{array}{|c|c|c|c|} \hline \text{green} & \text{green} & \text{green} & \text{green} \\ \hline \end{array}$$

$$x_3 \begin{array}{|c|c|c|c|} \hline \text{yellow} & \text{yellow} & \text{yellow} & \text{yellow} \\ \hline \end{array}$$

=

$$t_3 \begin{array}{|c|c|c|c|} \hline \text{orange} & \text{orange} & \text{orange} & \text{orange} \\ \hline \end{array}$$

+

$$x_3 \begin{array}{|c|c|c|c|} \hline \text{green} & \text{green} & \text{green} & \text{green} \\ \hline \end{array}$$

EMBEDDINGS

INPUT

je

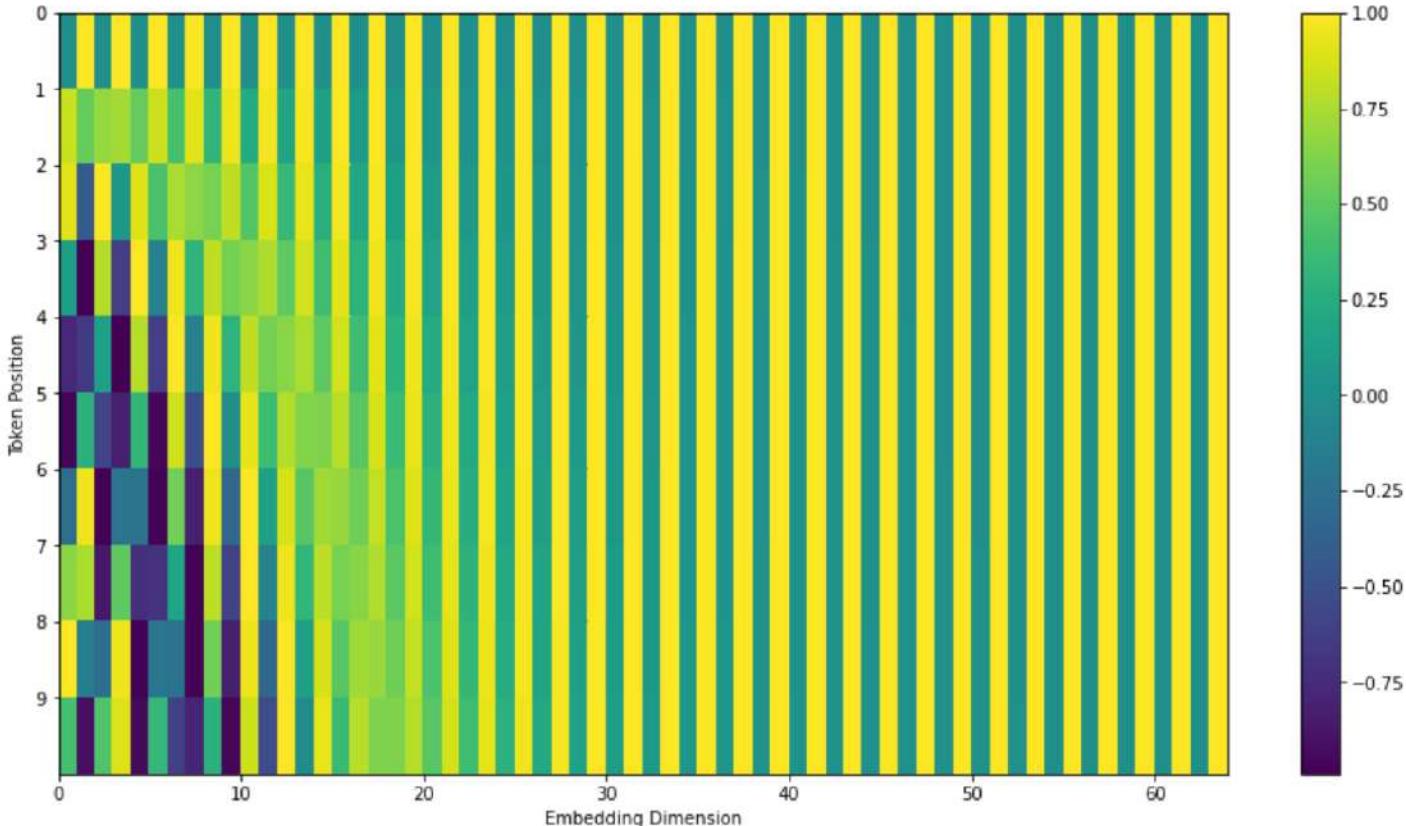
suis

étudiant

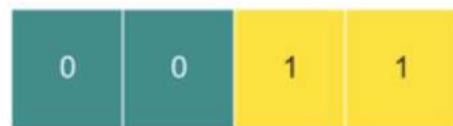
- Но если мы обрабатываем все слова одновременно, как нейронная сеть поймет, какое из них первое?
- Давайте закодируем номера слов в предложении

Positional encoding

- Пример positional encoding с игрушечным размером эмбеддинга 4
- Используем sin/cos чтобы посчитать позицию слова
- Как стрелки на часах и бинарный код вместе

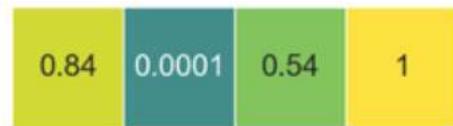
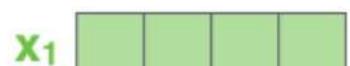


POSITIONAL
ENCODING

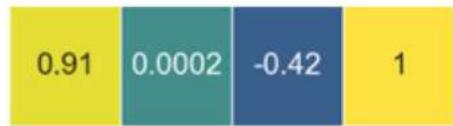


+

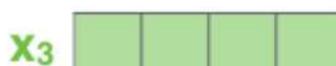
EMBEDDINGS



+



+



INPUT

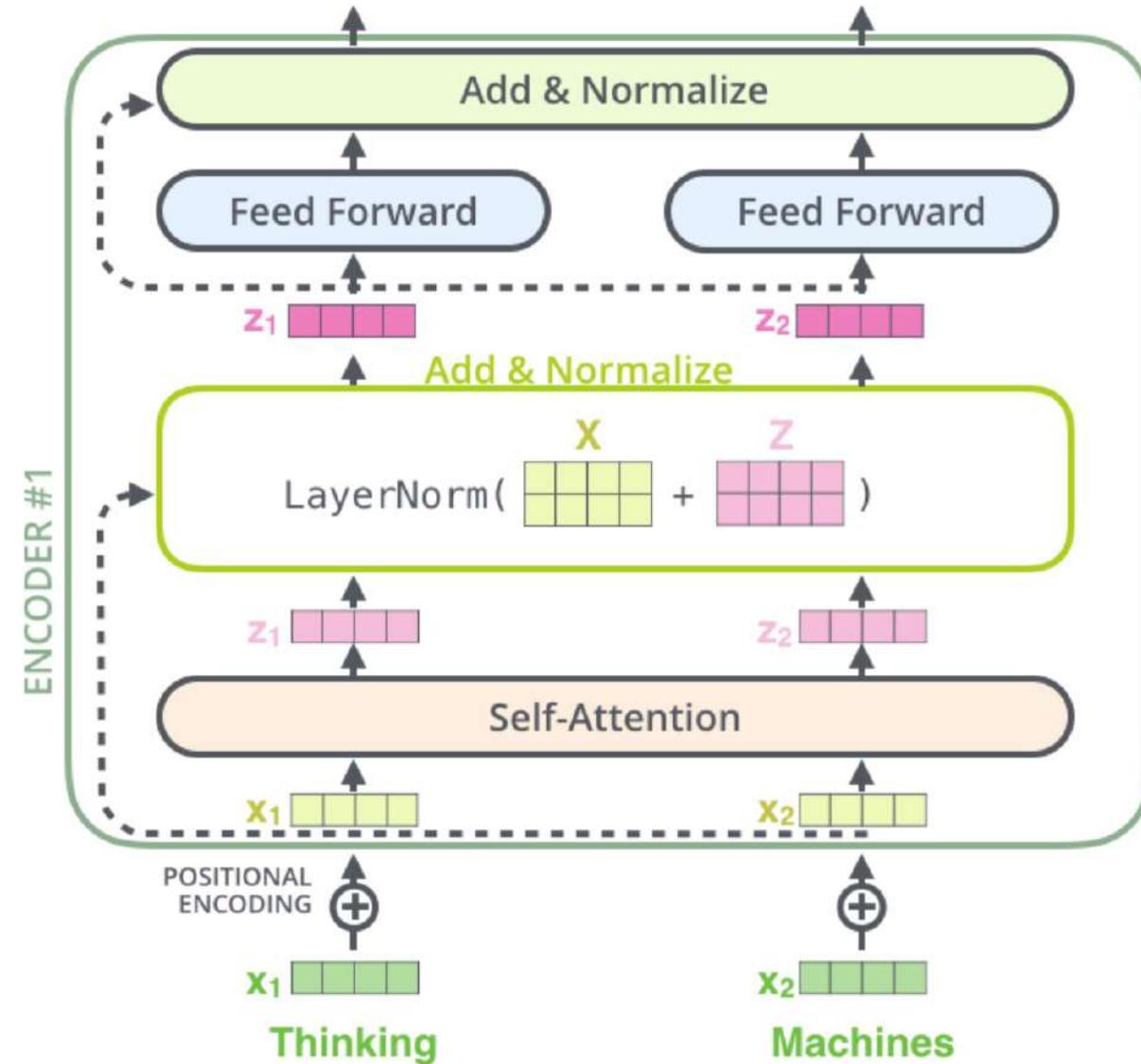
Je

suis

étudiant

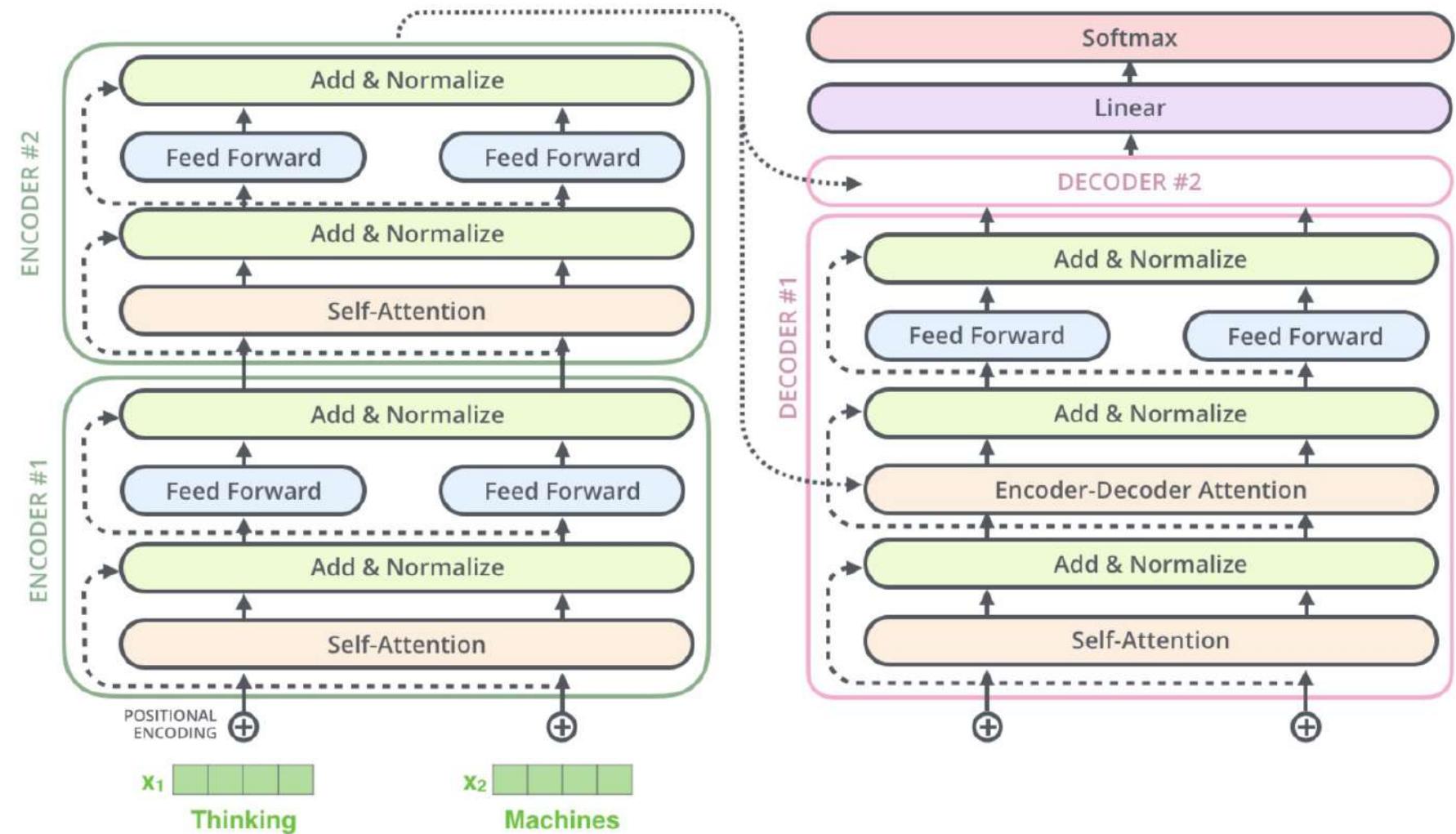
Add and Normalize

- Весь процесс вычислений в одном слое энкодера
- Все входные слова мы можем посчитать вместе и параллельно (как в CNN)
- В RNN мы и входной текст подаем по одному слову (формируем память)
- В трансформере нет скрытой памяти – новый сгенерированный текст это вся наша « память»



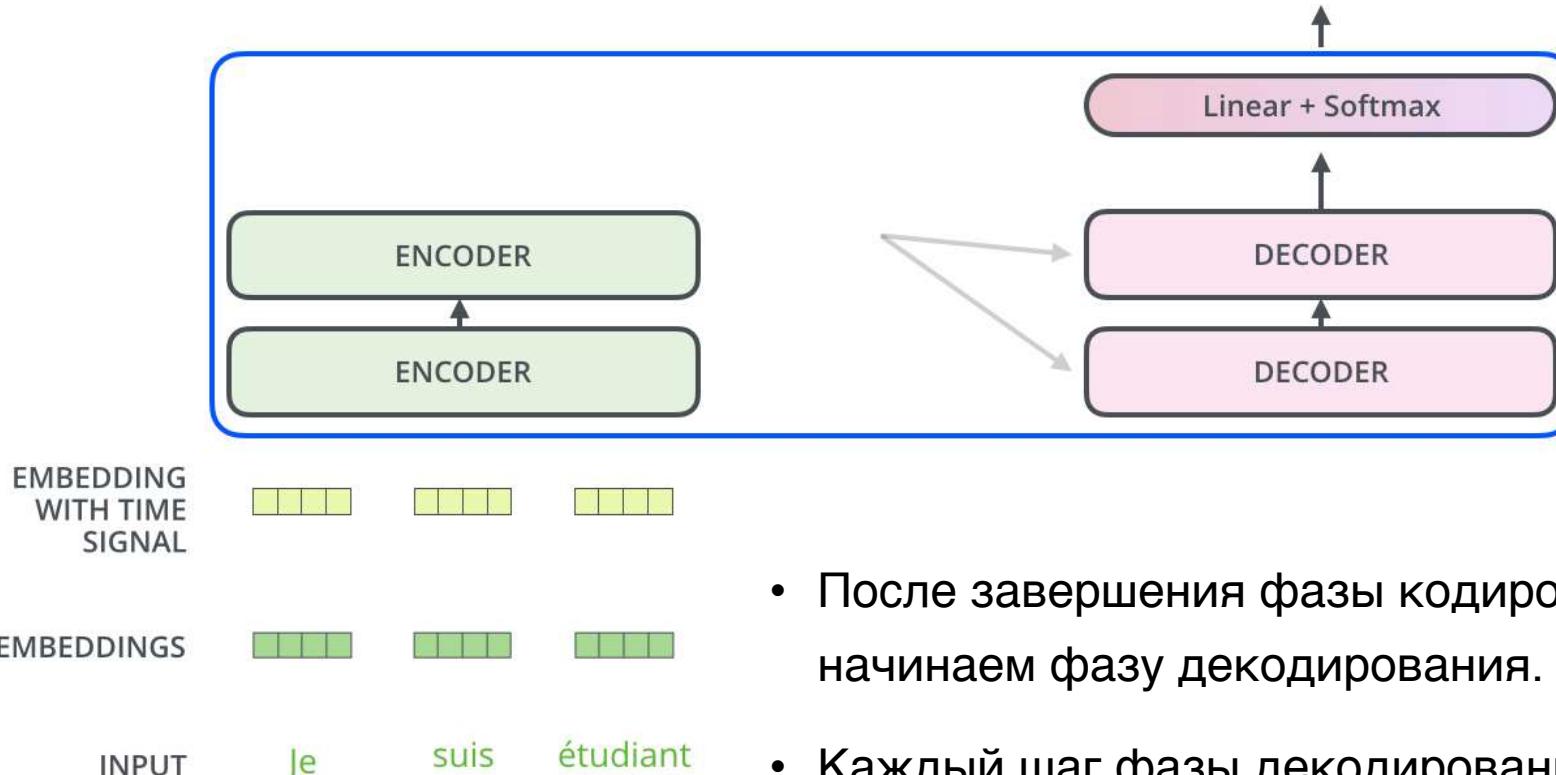
Encoder-decoder connection

- Вместе два слоя энкодера и два слоя декодера для двух входных слов
- Все входные данные мы уже имеем в тексте (как в авторегрессии)
- Нам НЕ нужно подавать текст только по одному слову как в RNN на вход нейросети
- Но генерация только по одному слову



Decoder

Decoding time step: 1 2 3 4 5 6



- После завершения фазы кодирования мы начинаем фазу декодирования.
- Каждый шаг фазы декодирования выводит элемент из выходной последовательности (в данном случае предложение перевода на английский язык)

Итоговое слово

- Модель знает 10,000 уникальных English слов (output vocabulary), которому мы обучили на training dataset
- Чтобы вывести нужное слово нам нужен вектор из 10,000 чисел – One-hot encoding, каждое число вероятность нужного слова.
- Это является выходом модели в Linear layer.

Which word in our vocabulary
is associated with this index?

Get the index of the cell
with the highest value
(argmax)

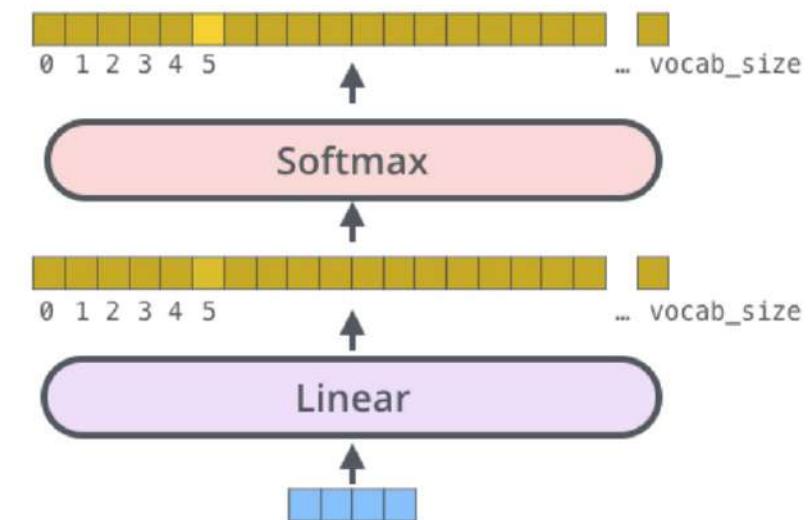
log_probs

logits

Decoder stack output

am

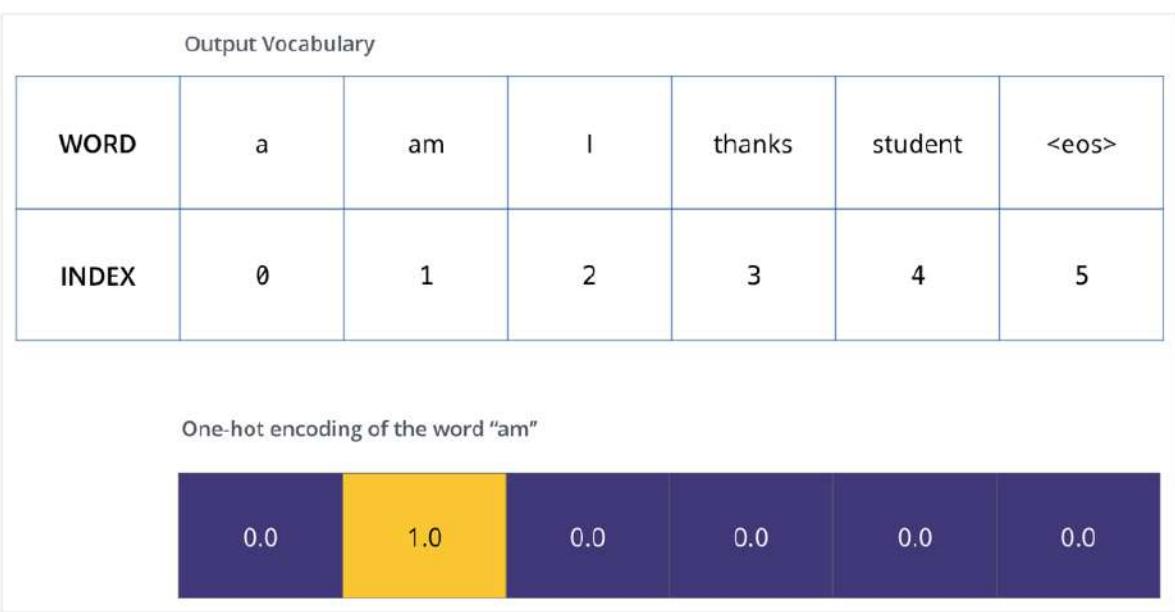
5



Model output

- Наша модель генерирует по очереди следующие слова
- В итоге у нас получается все предложение:

I am a student <eos>



Trained Model Outputs

Output Vocabulary: a am I thanks student <eos>

position #1	0.01	0.02	0.93	0.01	0.03	0.01
-------------	------	------	------	------	------	------

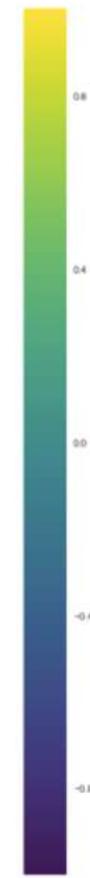
position #2	0.01	0.8	0.1	0.05	0.01	0.03
-------------	------	-----	-----	------	------	------

position #3	0.99	0.001	0.001	0.001	0.002	0.001
-------------	------	-------	-------	-------	-------	-------

position #4	0.001	0.002	0.001	0.02	0.94	0.01
-------------	-------	-------	-------	------	------	------

position #5	0.01	0.01	0.001	0.001	0.001	0.98
-------------	------	------	-------	-------	-------	------

a am I thanks student <eos>



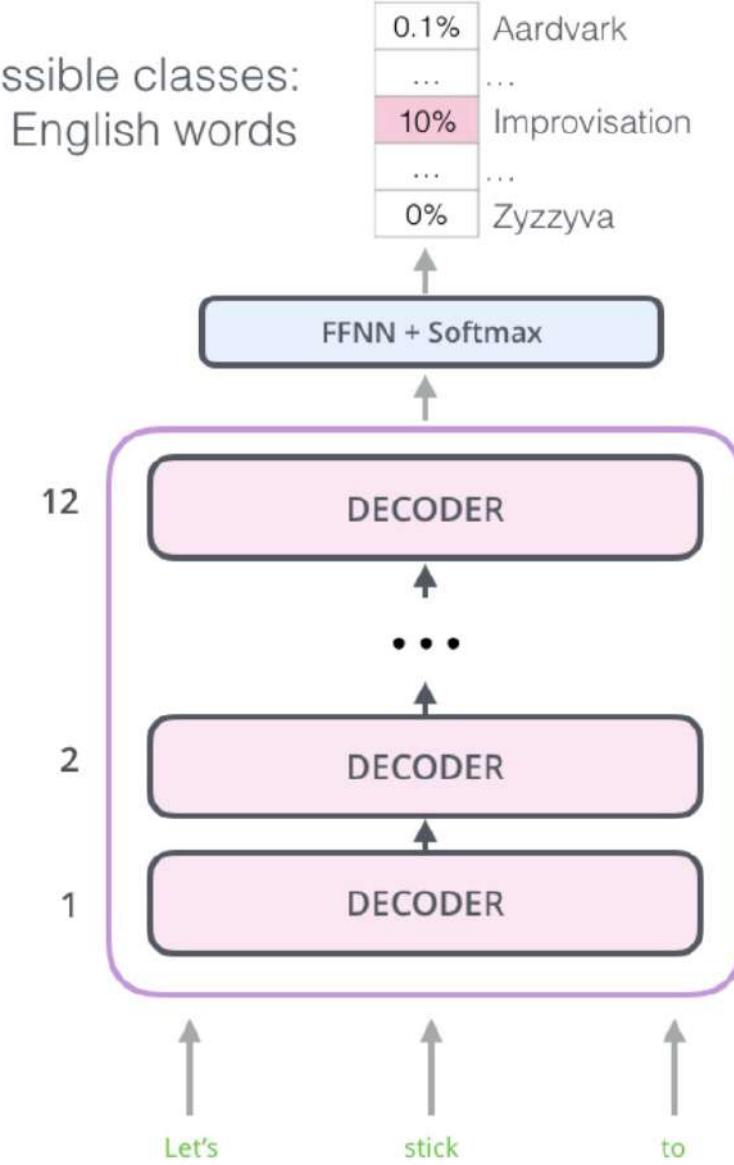
GPT

- GPT 2018 (117 млн параметров), 4.5 Гб текста
- GPT2 2019 (1.5 млрд параметров), обучен на основе Reddit (40Гб)
- GPT3 2020 (до 175 млрд параметров), 570 Гб текста
- GPT3.5 (InstructGPT) март 2022, обратная связь от человека
- В ноябре 2022 на основе InstructGPT был создан сервис ChatGPT
- Февраль 2023 представлена **открытая** модель Llama
- GPT4 март 2023
- Ноябрь 2024 компания Anthropic представила протокол MCP
- Январь 2025 Deepseek представила модель R1

GPT

- С помощью подобной структуры мы можем переходить к обучению модели для все той же задачи языкового моделирования: предсказать следующее слово, используя большой неразмеченный набор данных.
- Достаточно просто загрузить 7 тысяч книг и обучить на них модель.
- Книги для данного рода задач подходят отлично, т.к. они позволяют модели научиться находить связанные по смыслу фрагменты текста, даже если они значительно отстоят друг от друга
- Этого нельзя достигнуть, если обучать модель на твитах или новостных заметках.

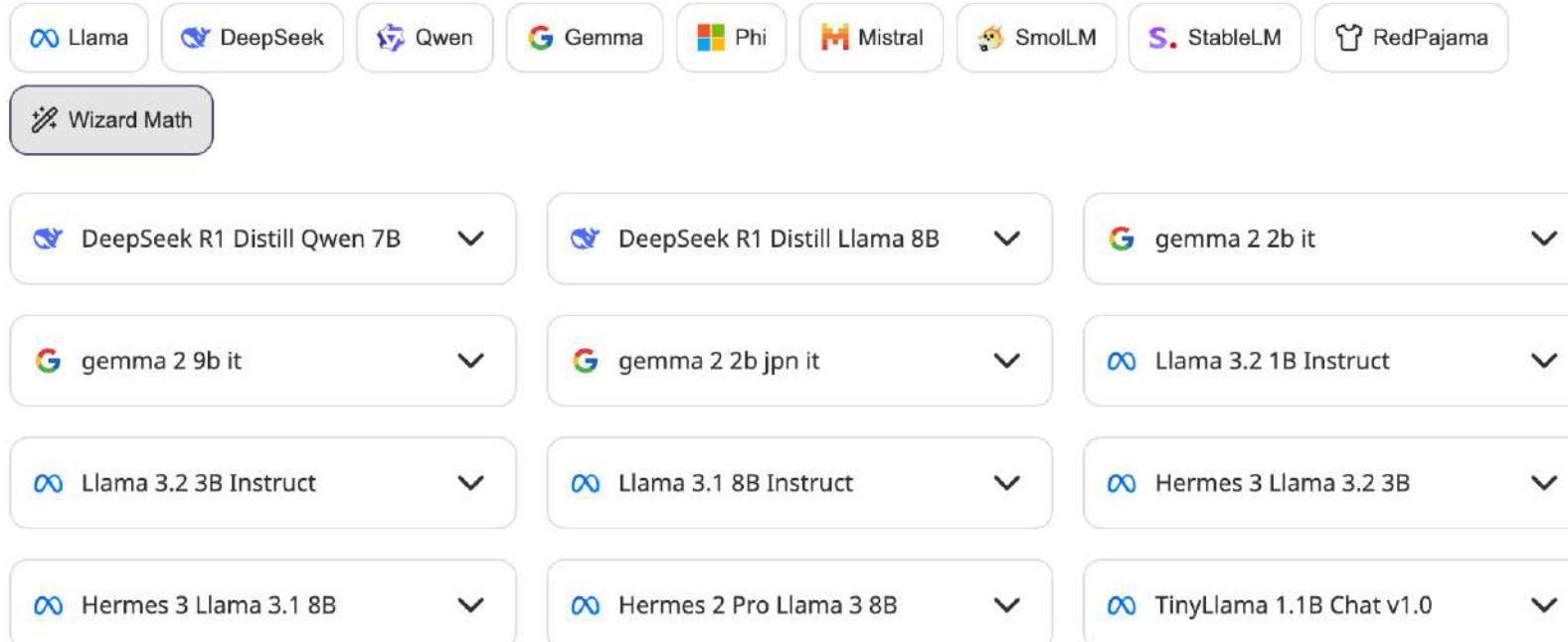
Possible classes:
All English words



Открытые LLM модели

Часть популярных моделей являются открытыми и каждый желающий, может их запустить у себя (Edge Device).

Список от mlc.ai:



и тд

Также почитать:

https://huggingface.co/spaces/open-llm-leaderboard/open_llm_leaderboard#/

Квантизация и дистилляция

Для запуска моделей на собственном устройстве (Edge Device) нам нужно уменьшить размер модели

Большие модели имеют миллиарды параметров, а значит требуют гигабайтов памяти

- Дистилляция – обучение более легковесной модели, ее учителем является большая модель с хорошими метриками
- Квантизация – использование 16 битных значений вместо 32-битных: меньшая точность, но меньший размер модели

WebLLM

- WebLLM – пакет JavaScript, который позволяет запускать LLM прямо в браузере
- Использует WebGPU
- Вам требуется GoogleChrome
- Можете использовать разные модели, настроить температуру
- Температура – параметр вариативности ответа: низкая температура оставляет только токены с высокими вероятностями

Model Type WebLLM Models

Модель Llama-3.2-1B-Instruct-q4f32_1-MLC (Meta)

Context Window Length
The maximum number of tokens for the context window 2K

Температура
Чем выше значение, тем более случайный вывод 0.1

Top P
Do not alter this value together with temperature 0.9

Максимальное количество токенов
Максимальная длина вводных и генерируемых токенов 2000

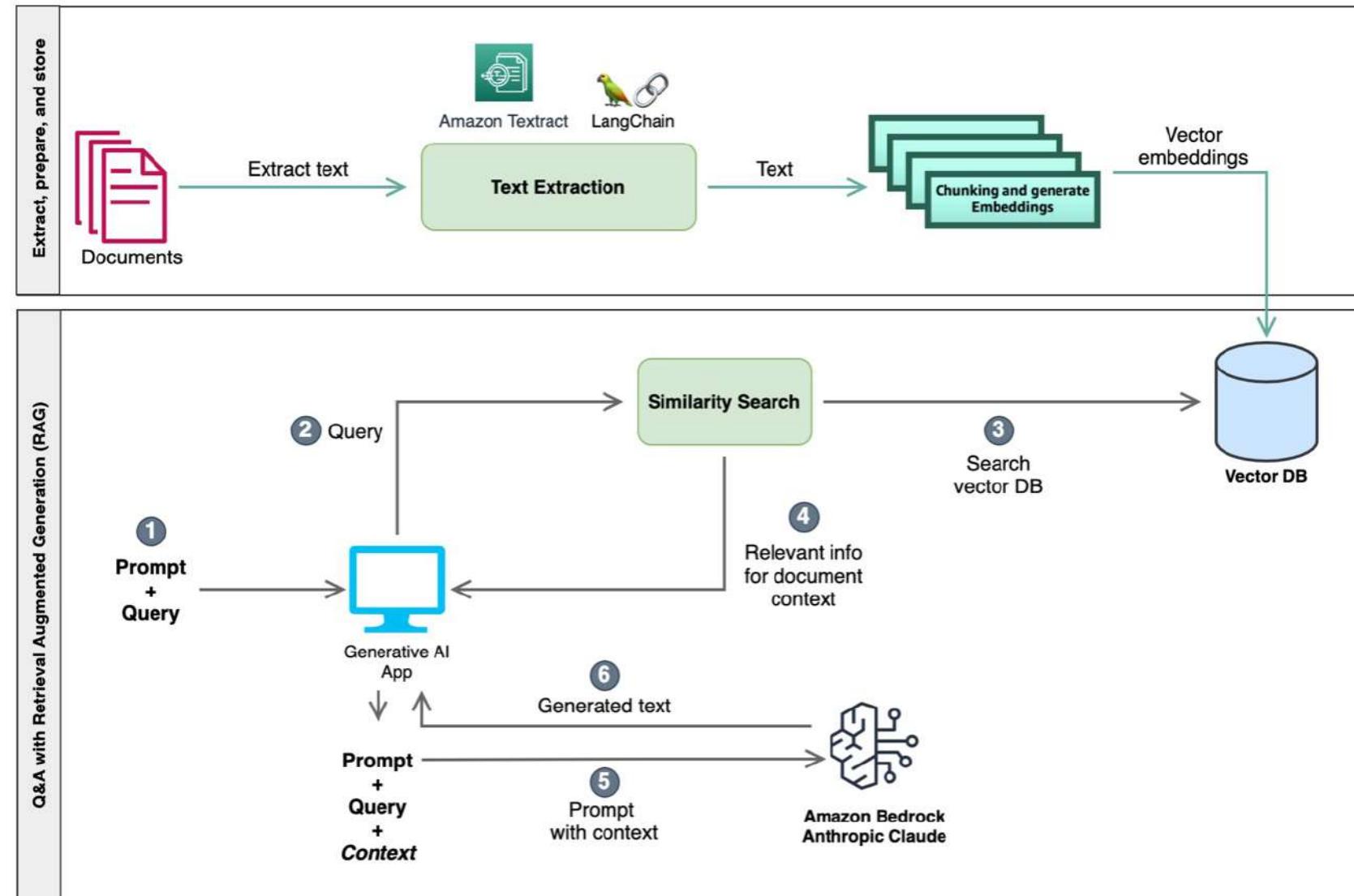
Штраф за повторения
Чем выше значение, тем больше вероятность общения на новые темы 0.3

Штраф за частоту
Большее значение снижает вероятность повторения одной и той же строки 0.5

<https://chat.webllm.ai/>

Retrieval Augmented Generation (RAG)

- LLM часто генерируют галюцинации
- Лучше добавить в запрос подходящие данные (аналог Google)
- BERT превращает запрос и документы в векторы
- Мы сравниваем векторы и похожие подаем вместе с запросом в LLM



Лекция 6

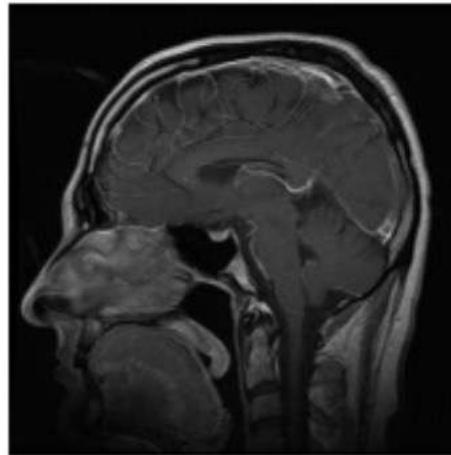
Сегментация и LiDAR

Разработка нейросетевых систем

Канев Антон Игоревич

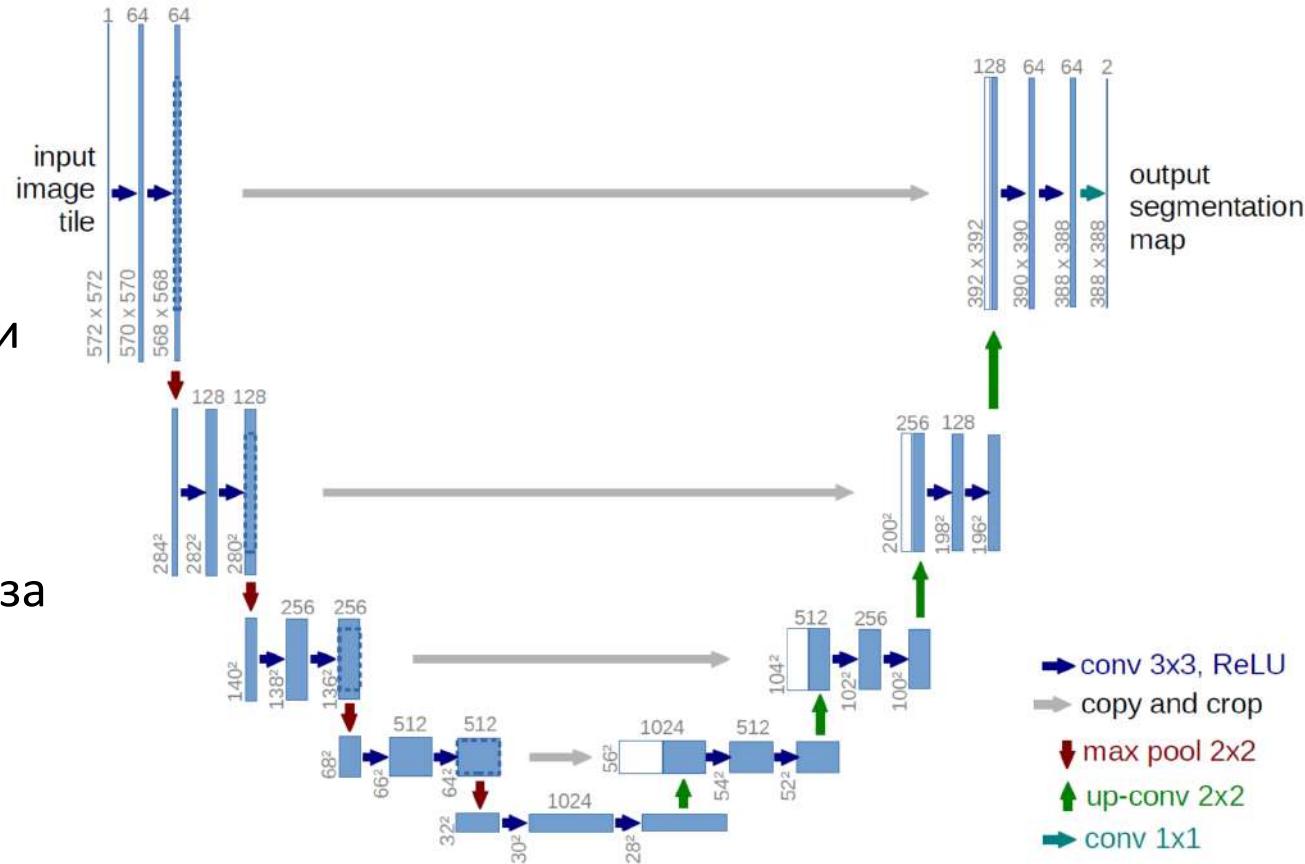
Сегментация

- Сегментация изображений является задачей разбиения цифрового изображения на одну или несколько областей, представляющих интерес.
- В модели для сегментации нет полносвязных слоёв, как в сетях для классификации
- Мы используем её для сегментации и в результате хотим получить картинку-маску, размер которой зависит от размера входного изображения.



U-Net

- Архитектура состоит из сужающегося пути и расширяющегося пути.
- Сужающийся путь — типичная архитектура сверточной нейронной сети. Он состоит из повторного применения двух сверток 3×3 , за которыми следуют ReLU и max pool 2×2 .
- Размер слоев становится все меньше и меньше для изучения более частных признаков, известный как «понижающая выборка».
- Сеть включает 4 соединения с пропуском — после каждой транспонированной свертки (или «up-conv») на пути повышающей дискретизации результирующая карта объектов объединяется с одной из пути понижающей дискретизации.



- pool/свертка 2×2 уменьшает/увеличивает количество каналов свойств;
- объединение с соответствующей обрезанной картой свойств из стягивающегося пути;
- две 3×3 свертки, за которыми следует ReLU.
- свертка 1×1 для сопоставления 64-мерного вектора с нужным количеством классов

Dice

- Dice коэффициент показывает попиксельное соотношение между прогнозируемой маской и соответствующей ей истиной
- Коэффициент Dice — это двойная площадь их пересечения, деленная на общее количество пикселей в обоих изображениях.
- Dice принимает значения от 0 до 1 (полное совпадение)

$$DC = \frac{2TP}{2TP + FP + FN} = \frac{2|X * Y|}{|X| + |Y|}$$

Ансамбли моделей

Усреднение прогнозов моделей

- Самый простой способ объединения предсказания подмоделей - это рассчитать их среднее значение. Также можно добавить веса, чтобы сделать более ощутимым вклад удачных моделей.

Stacking Ensemble

- Мы можем обучить мета-ученика, который будет сочетать прогнозы из подмоделей и в идеале делать предсказания точнее, чем любая отдельная подмодель.
- В качестве мета-ученика можно использовать линейную регрессию или нейронную сеть. Во втором случае можно рассматривать стековый ансамбль как единую большую модель (multi-headed model)
- Подмодели могут быть встроены в более крупную многоголовую нейронную сеть, которая затем учится, как лучше всего комбинировать прогнозы из каждой входной подмодели.

Keras. Сегментация

- Входному слою каждой подмодели нужно предоставить свои входные данные
- Затем скрытый слой будет подбирать веса для интерпретации этого «входа»
- Выходной слой (сверточный+активация sigmoid) делать свой собственный вероятностный прогноз.

```
model1 = tf.keras.models.load_model(local_download_path + 'files/model186_dice.h5', compile=False)
model2 = tf.keras.models.load_model(local_download_path + 'files/model230_bc.h5', compile=False)
model3 = tf.keras.models.load_model(local_download_path + 'files/model230_tversky.h5', compile=False)
model4 = tf.keras.models.load_model(local_download_path + 'files/model230_dice_bc.h5', compile=False)
model5 = tf.keras.models.load_model(local_download_path + 'files/model_dice_wind.h5', compile=False)
```

```
members = [model1, model2]

from keras.layers import concatenate

ensemble_visible = [model.input for model in members]
ensemble_outputs = [model.output for model in members]
x = concatenate(ensemble_outputs)

x = Conv2D(16, (3, 3), padding="same")(x)
#x = Dropout(0.5)(x)
x = BatchNormalization()(x)
x = Activation("relu")(x)

x = Conv2D(1, (1, 1), padding="same")(x)
x = Activation("sigmoid")(x)
```

Keras. Обучение ансамблей

- Keras – надстройка над tensorflow для обучения нейронных сетей
- Обучение на Keras схоже с PyTorch
- Сами циклы обучения теперь скрыты в функции fit()

```
lr = 1e-3

opt = tf.keras.optimizers.Adam(learning_rate=lr)
metrics = [dice_coefficient]

ensemble_model.compile(loss=dice_loss, optimizer=opt, metrics=metrics)
```

```
batch = 8
epochs = 20

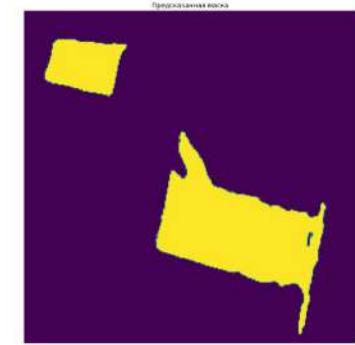
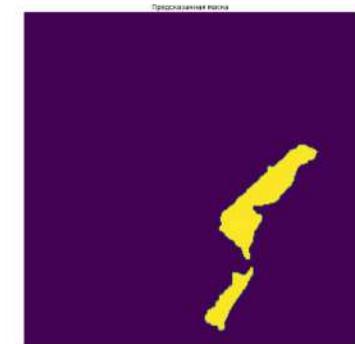
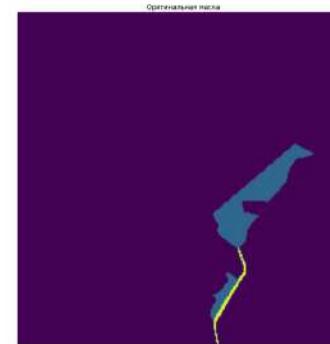
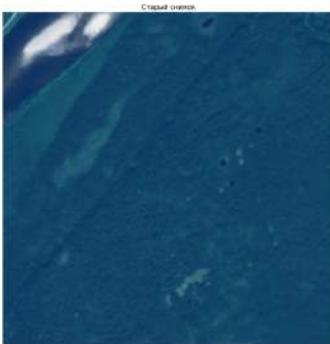
validSize = int(len(trainTiles)*0.15)

train_generator = DataGenerator(trainTiles[:-validSize],batch_size=batch,num_models=2)
valid_generator = DataGenerator(trainTiles[-validSize:],batch_size=batch,num_models=2)

hist = ensemble_model.fit(train_generator,
                          steps_per_epoch=len(trainTiles[:-validSize])//batch,
                          validation_data=valid_generator,
                          epochs=epochs,
                          validation_steps=len(trainTiles[-validSize:])//batch,
                          # callbacks=callbacks
)
```

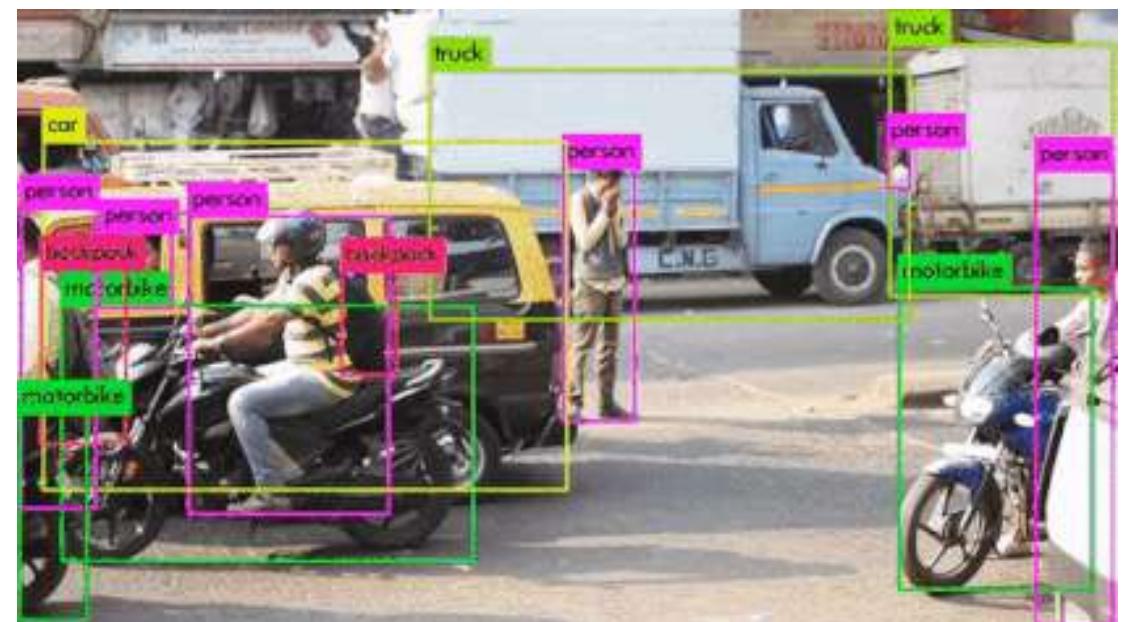
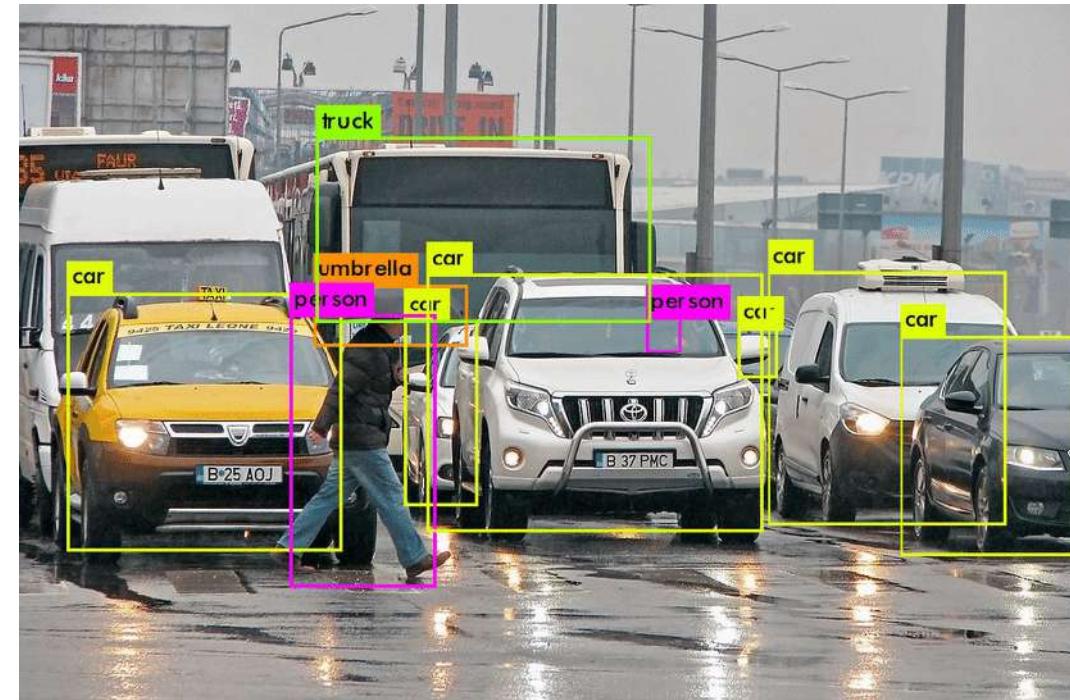
Сегментация результаты

- Сравним маску, полученную моделью (справа), и оригинальную маску (слева)
- Для наглядности выведем старый и новый снимки



Обнаружение объектов

- **Object Detection** (обнаружение объектов) – определение объекта на изображении или в видео потоке
- Для этого используются различные модели, которые разделяются на «двухуровневые», такие как R-CNN, fast R-CNN и faster R-CNN, и «одноуровневые», такие как YOLO



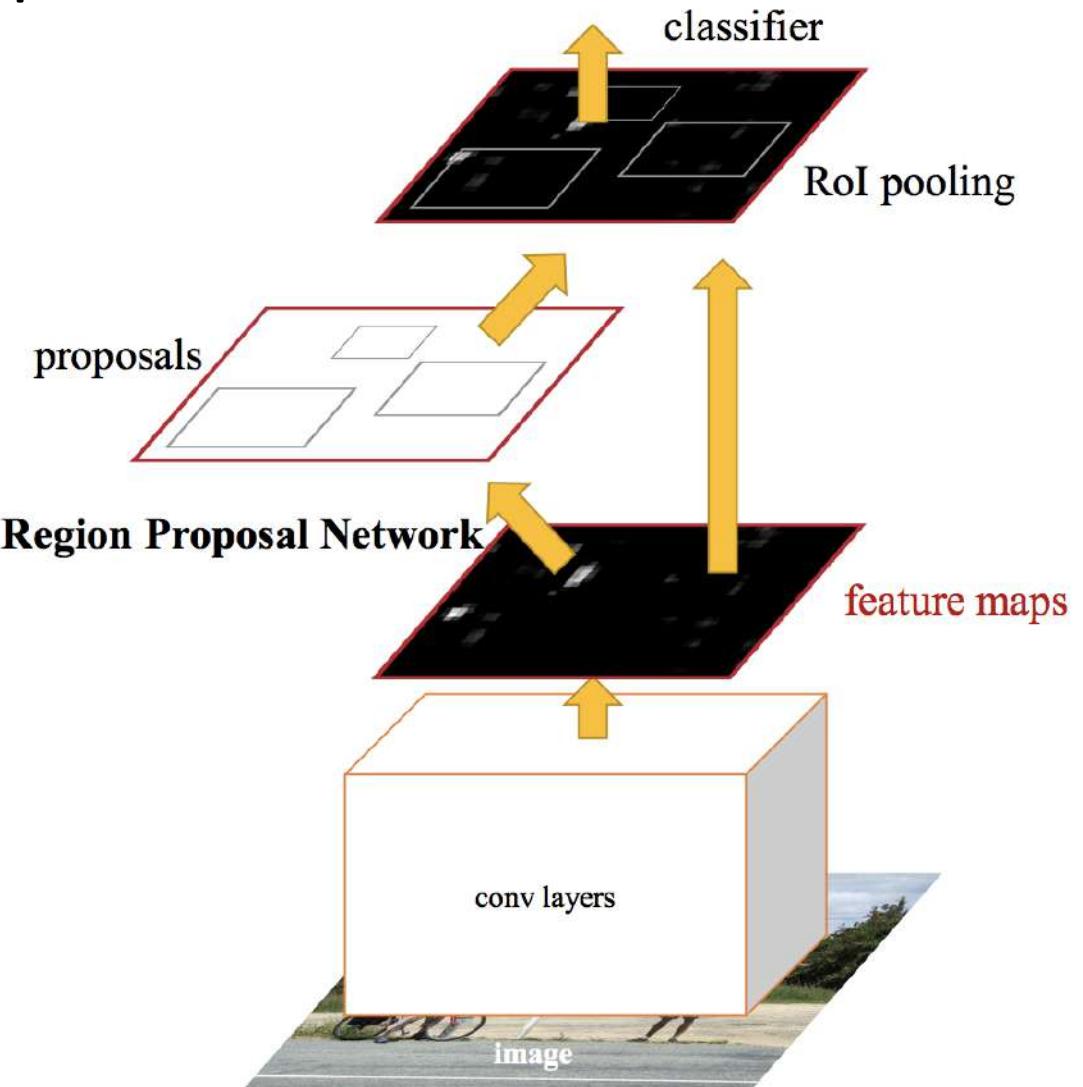
Модели Object Detection

Faster R-CNN:

- Подается картинка/кадр на вход
- Кадр прогоняется через CNN для формирования feature maps
- Отдельной нейронной сетью определяются регионы с высокой вероятностью нахождения в них объектов
- Дальше эти регионы с помощью ROI pooling сжимаются и подаются в нейронную сеть, определяющую класс объекта в регионах

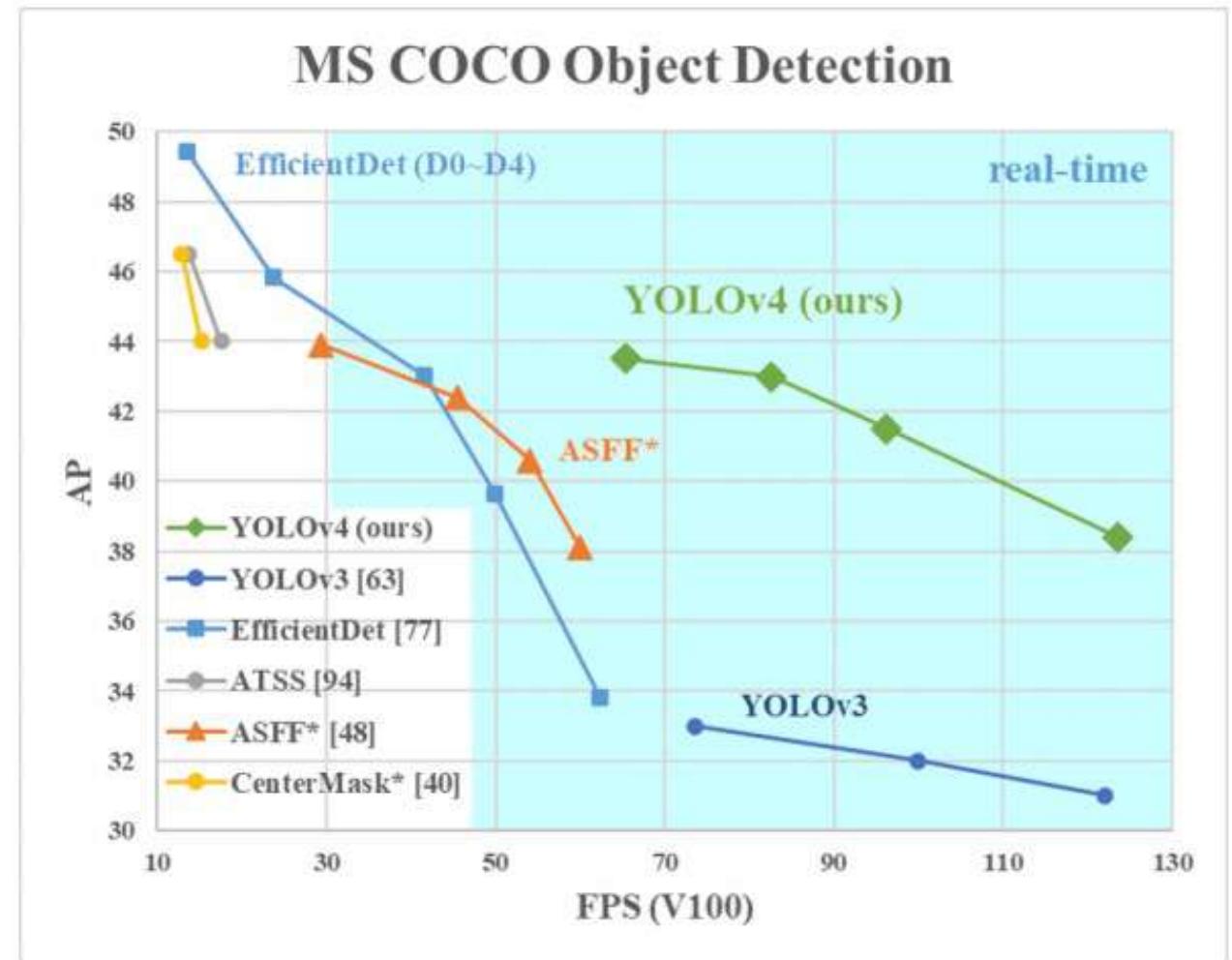
<https://arxiv.org/pdf/1311.2524v4.pdf>

R-CNN



YOLO

- YOLO (You Only Look Once) смотрит на картинку один раз, и за этот один прогон картинки через одну нейронную сеть делает все необходимые определения объектов
- За счет этого повышается производительность модели
- Существуют разные версии модели (v3, v4, v7 и тд)

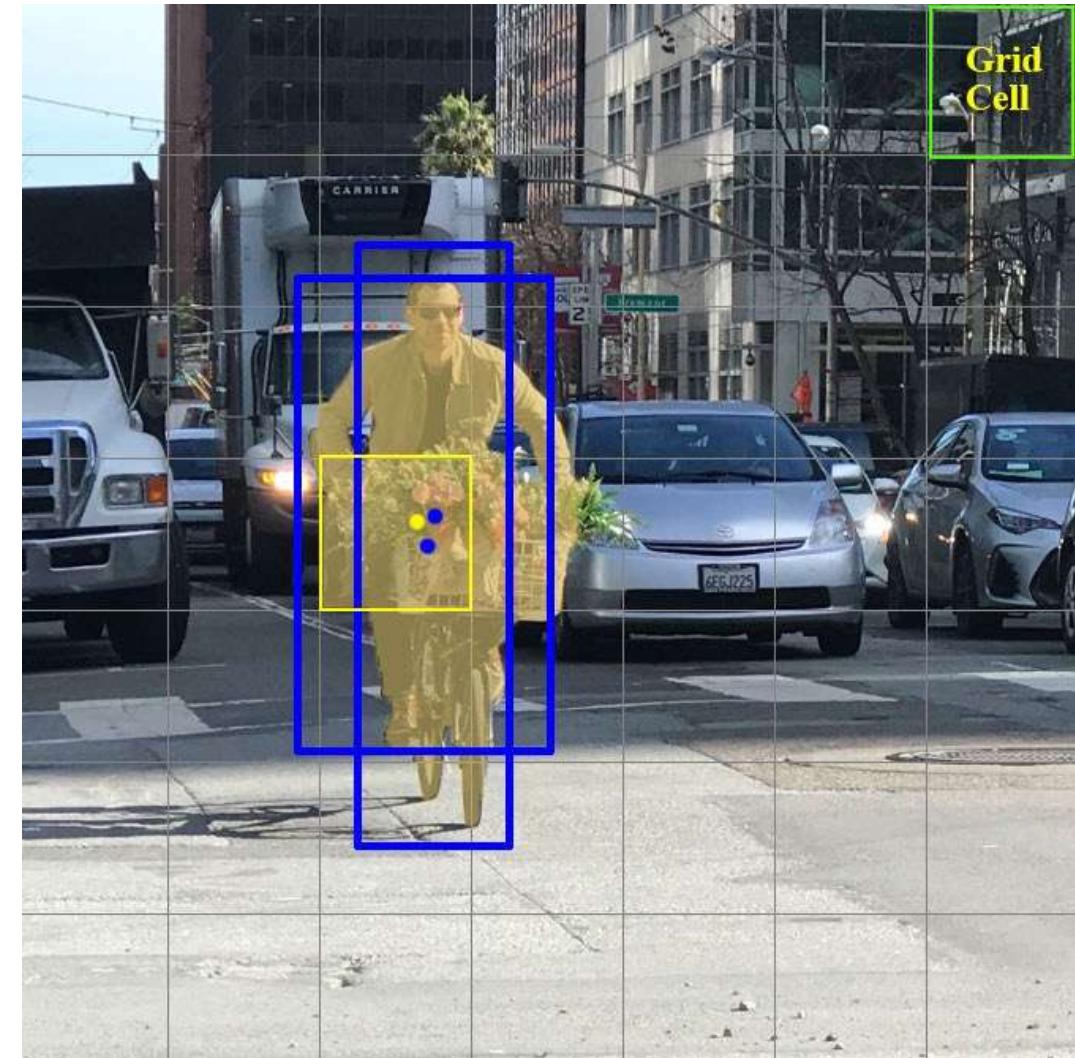
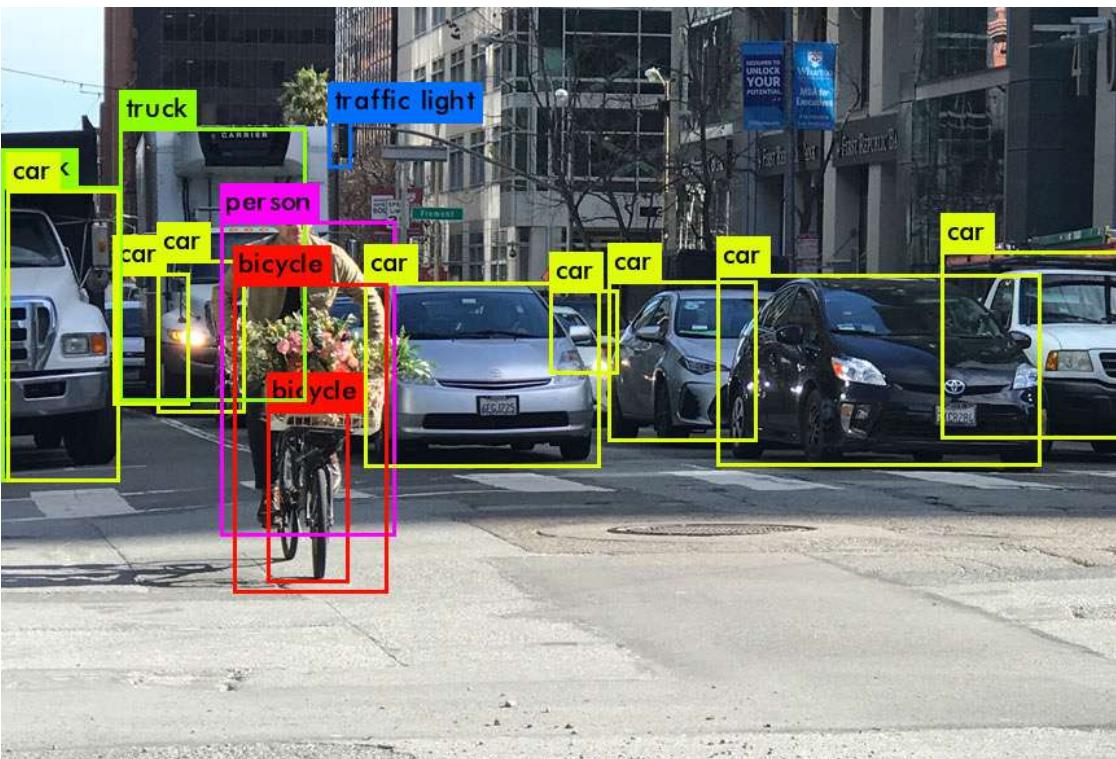


YOLOv3

<https://pjreddie.com/media/files/papers/YOLOv3.pdf>

YOLO

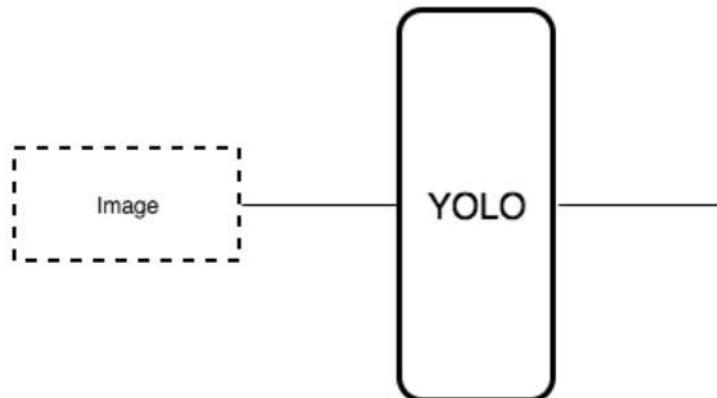
- Каждая клетка grid cell является «якорем», к которому прикрепляются bounding boxes. Это основа идеи YOLO
- Вокруг клетки рисуются несколько прямоугольников для определения объекта (непонятно, какой будет наиболее подходящим), и их позиции, ширина и высота вычисляются относительно центра этой клетки.



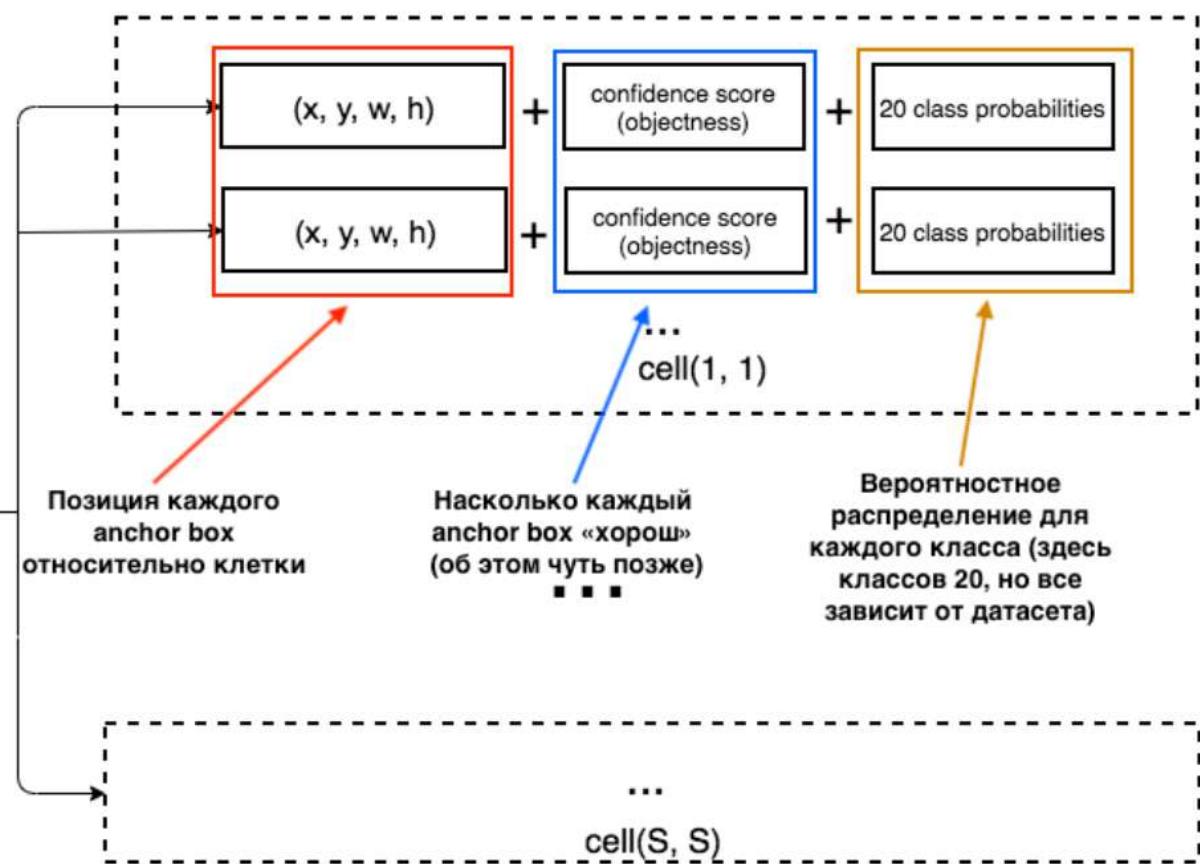
Выход YOLO

Нам нужно понять две принципиальные вещи:

- Какой из anchor boxes, из 3 нарисованных вокруг клетки, нам подходит больше всего и как его можно немного подправить для того, чтобы он хорошо вписывал в себя объект

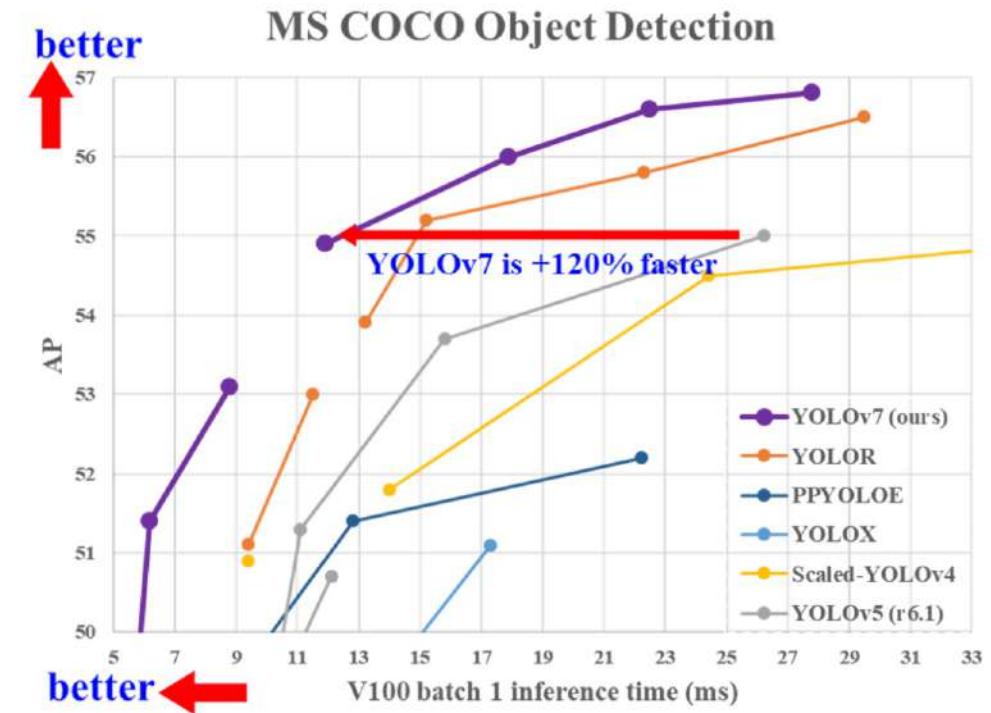
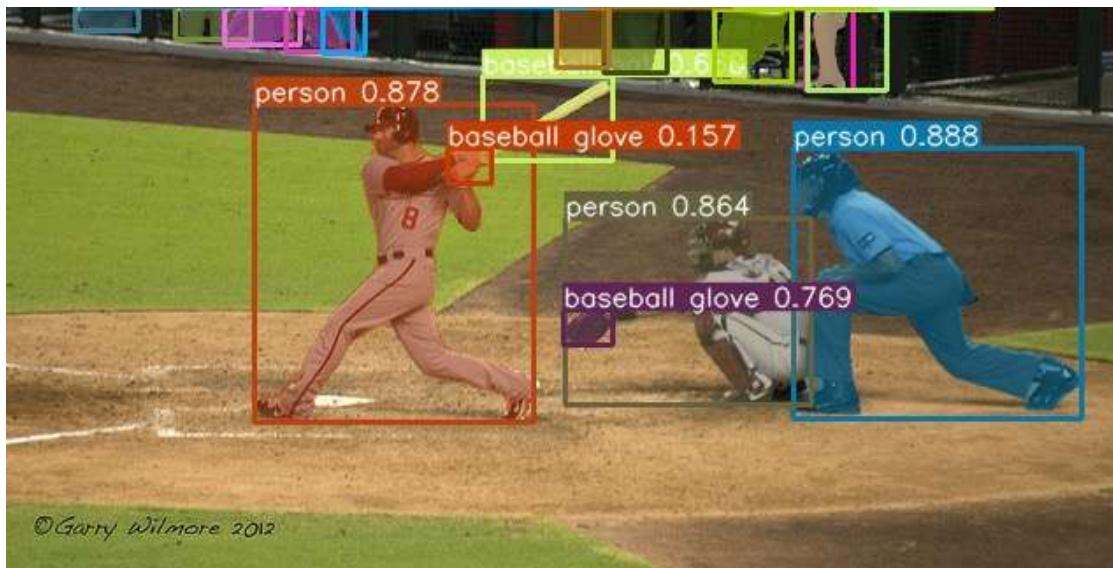


- Какой объект находится внутри этого anchor box и есть ли он вообще



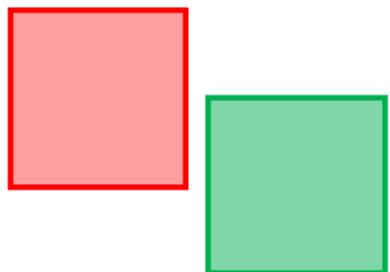
YOLOv7

- Помимо object detection модель YOLO используется для оценки положения или сегментации

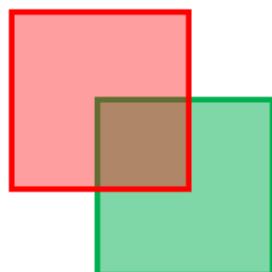


IoU

- Метрика Intersection over Union (IoU), также известная как Jaccard index
- Число от 0 до 1, показывающее, насколько у двух объектов (эталонного и текущего) совпадает общая часть



IoU = 0



IoU = 0.142

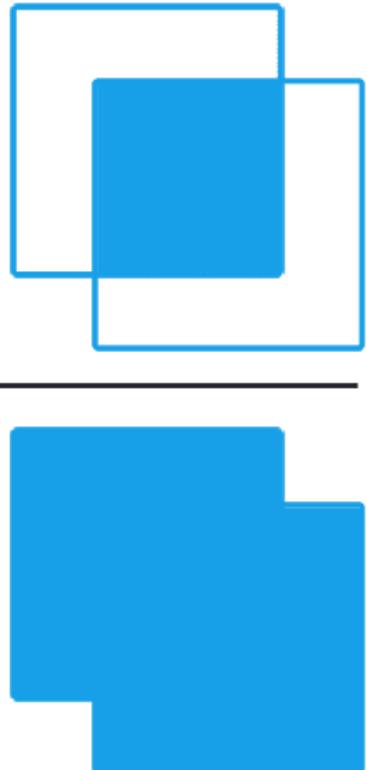


IoU = 0.333



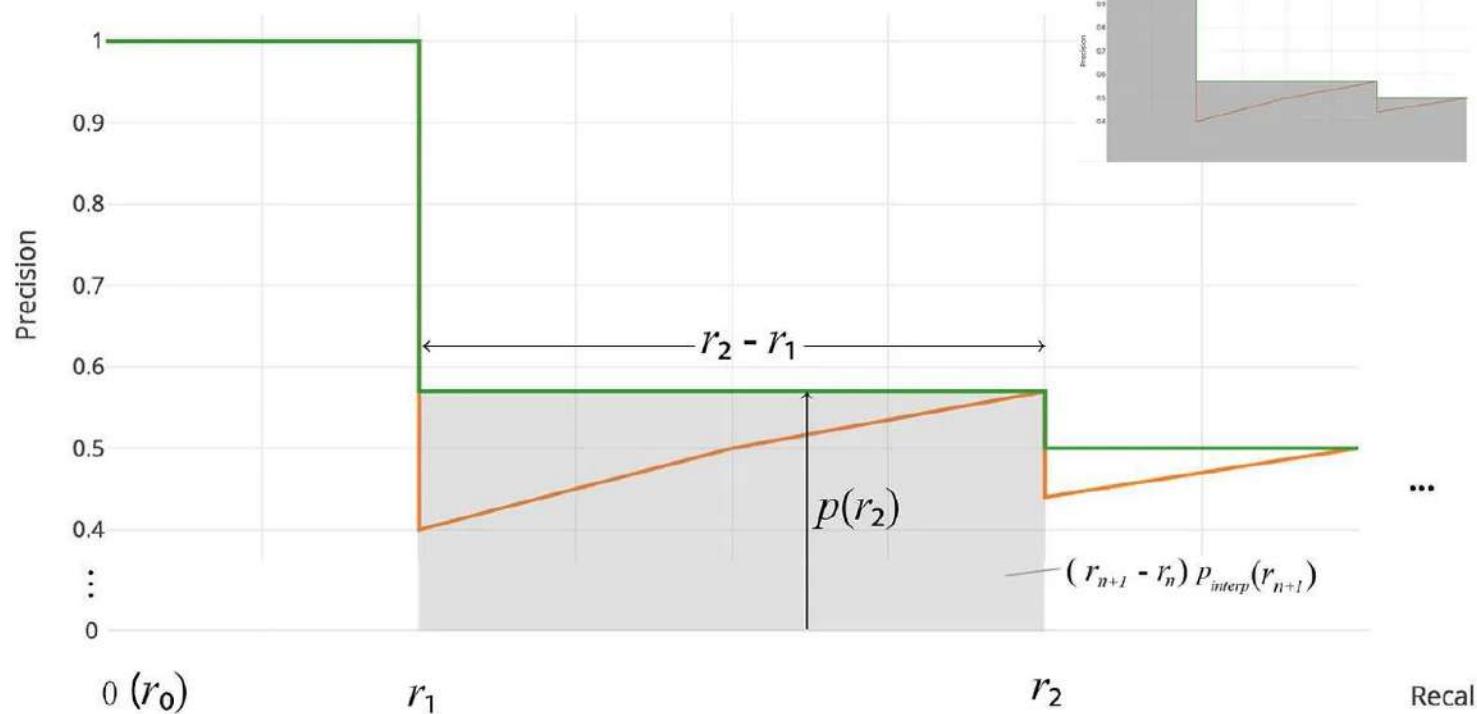
IoU = 1

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$



AP

- AP (Average precision) – популярная метрика для оценки точности моделей обнаружения объектов



Rank	Correct?	Precision	Recall
1	True	1.0 ↑	0.2 ↑
2	True	1.0 –	0.4 ↑
3	False	0.67 ↓	0.4 –
4	False	0.5 ↓	0.4 –
5	False	0.4 ↓	0.4 –
6	True	0.5 ↑	0.6 ↑
7	True	0.57 ↑	0.8 ↑

- Precision и Recall связаны друг с другом: чем больше одна, тем обычно меньше другая метрика

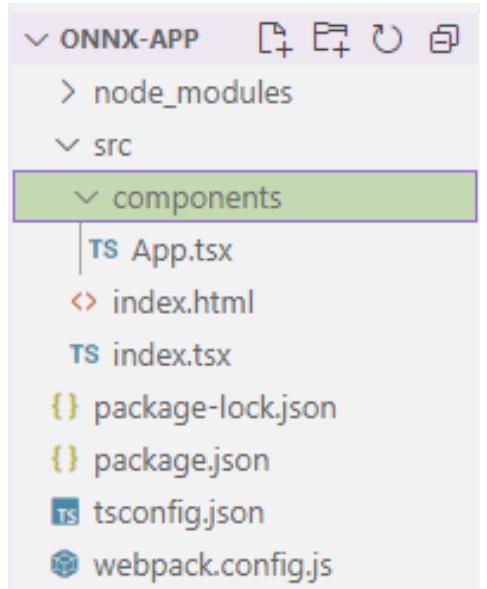
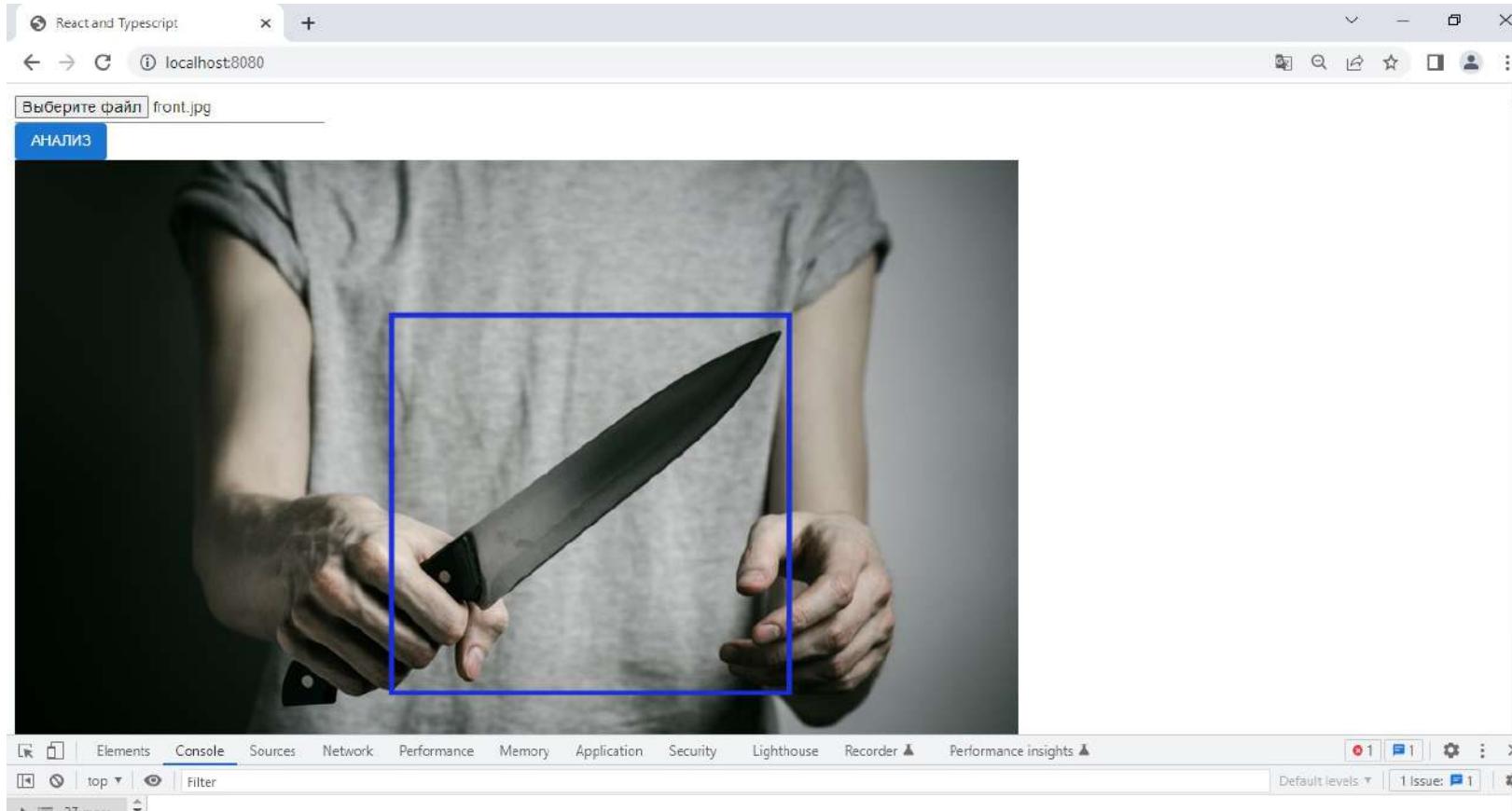
Разметка изображений

- Cvat.ai – можно развернуть локально и работать совместно
- ImageJ – устанавливается локально
- Roboflow – поддерживается аугментация, разметка
- ij.imjoy.io
- VGG Image Annotator



<https://www.cvat.ai>

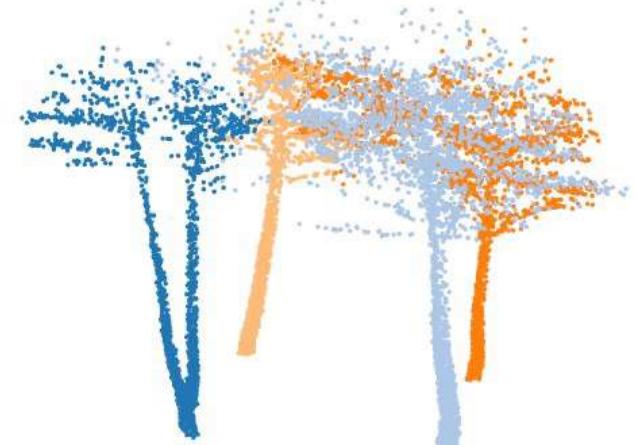
Приложение React



- Для использования обученной модели Yolo (.onnx) вы создаете приложение React
- Это SPA приложение, которое полностью выполняется в браузере на языке JavaScript

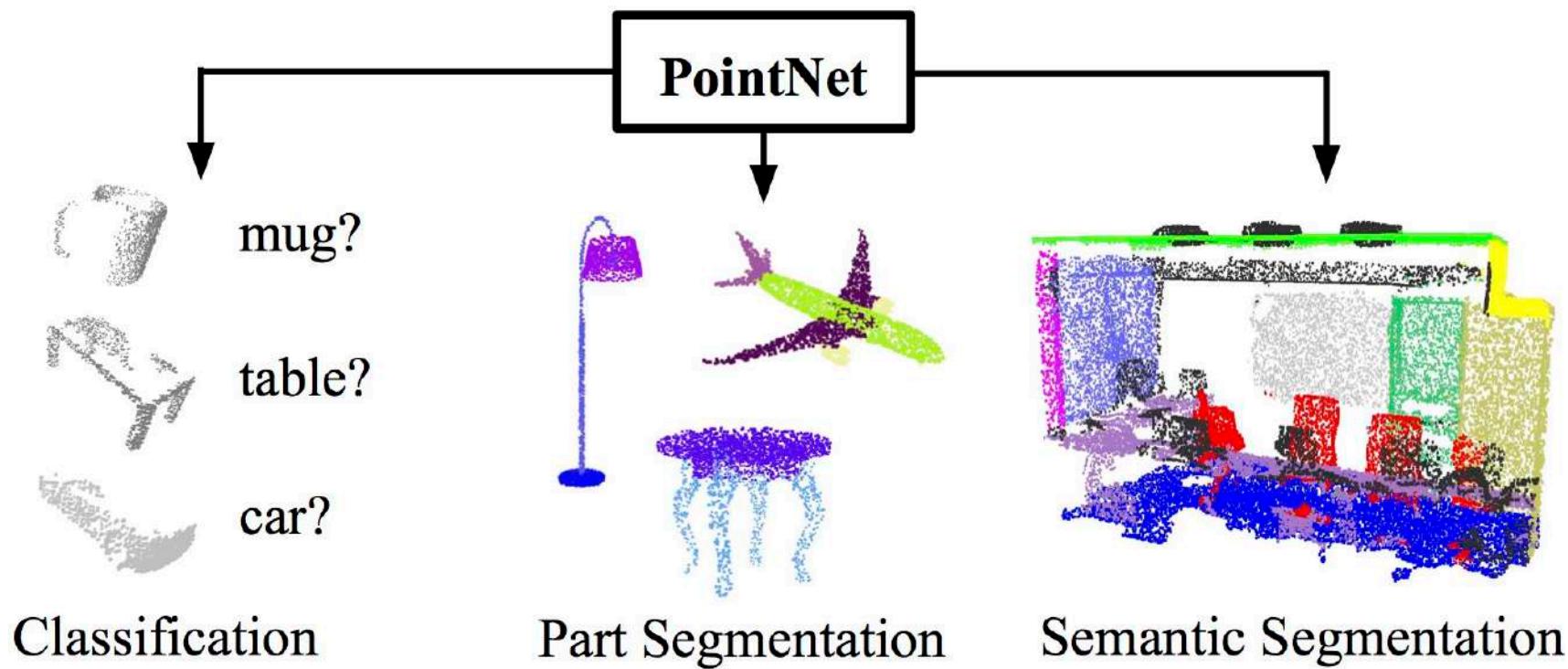
LiDAR

- Облако точек - это самый простой способ представления различных объектов в виде неупорядоченного набора точек в трехмерной плоскости. Такие данные можно получить с помощью сканирования предметов или их структуры с помощью 3D-датчиков, например LiDAR.
- Качественные облака точек с высокой точностью измерения позволяют представить цифровую версию реального мира.
- Основная проблема работы с облаком точек заключается в том, что типичная сверточная архитектура требует упорядоченный формат входных данных (например, изображение).
- Поскольку облако точек не является таким, общепринятые подходы заключаются в преобразовании данных в обычную 3D-voxельную сетку или проекцию.



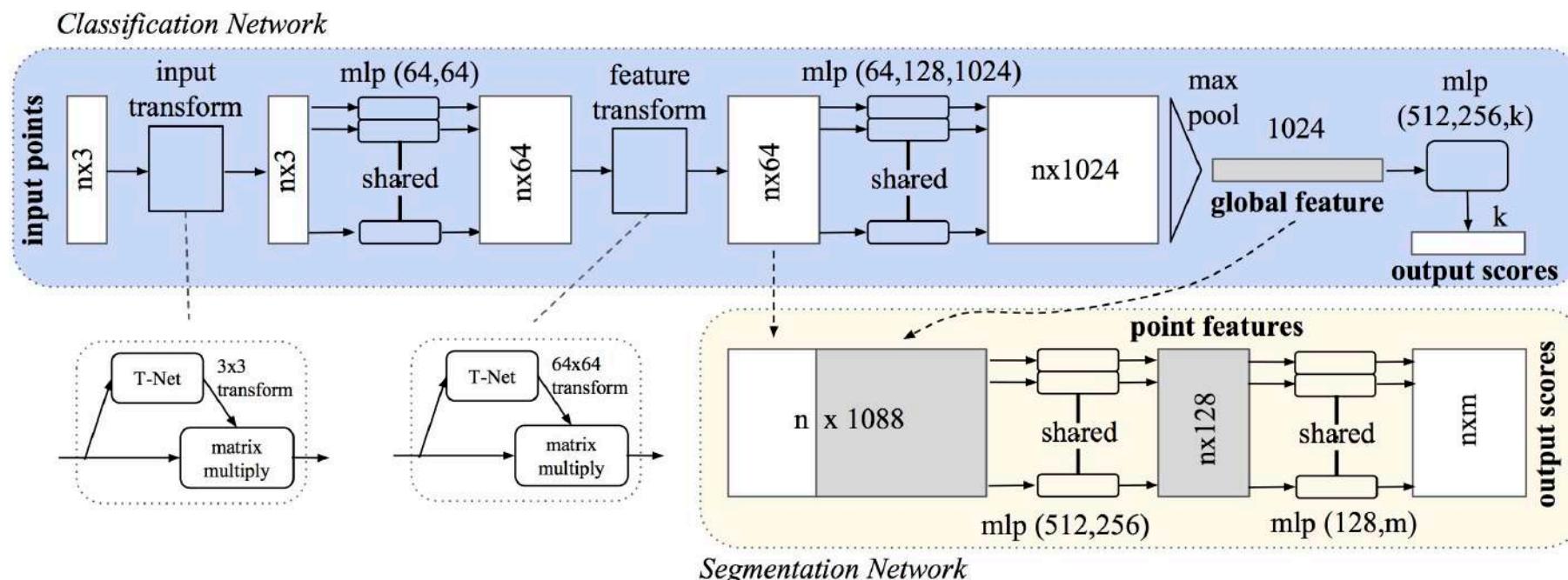
PointNet. Применение

- PointNet использует неупорядоченные облака точек и может выполнять классификацию и сегментацию объектов, а также семантический анализ сцены.



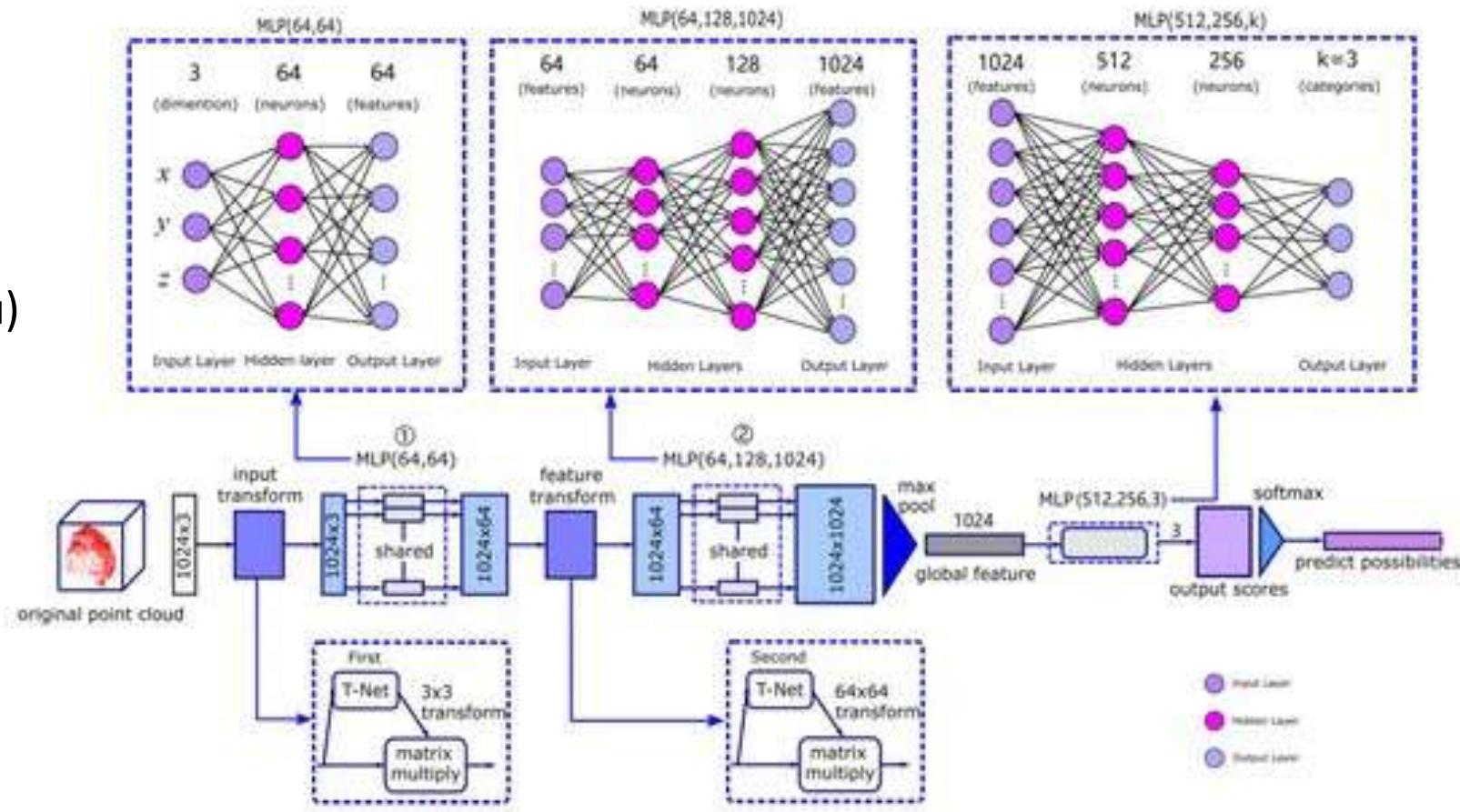
PointNet. Архитектура

- В сети есть 3 ключевых модуля: слой Max Pooling, принимающий n векторов входных данных и выводящий новый вектор, две сети трансформации с многослойным персепtronом (MLP) с размерами (64,64) и (64,128,1024) и две сети для предсказания с обученной матрицей T-Net.
- PointNet изучает характеристики каждой точки с помощью MLP и объединяет все характеристики с помощью симметричной функции для классификации объектов и их сегментации на части.



Shared MLP

- Shared MLP – это несколько копий (серые прямоугольники) одной и той же полносвязной сети с одинаковыми весами
- Количество копий равно количеству точек n (1024)
- Каждый слой такой сети мы реализуем с помощью отдельного сверточного слоя CONV-1D ($\text{kernel}=1$)
- Количество нейронов (64-1024) мы указываем через количество каналов



Лабораторная LiDAR

```
inputs = keras.Input(shape=(NUM_POINTS, 3))

x = tnet(inputs, 3)
x = conv_bn(x, 32)
x = conv_bn(x, 32)
x = tnet(x, 32)
x = conv_bn(x, 32)
x = conv_bn(x, 64)
x = conv_bn(x, 512)
x = layers.GlobalMaxPooling1D()(x)
x = dense_bn(x, 256)
x = layers.Dropout(0.3)(x)
x = dense_bn(x, 128)
x = layers.Dropout(0.3)(x)

outputs = layers.Dense(NUM_CLASSES, activation="softmax")(x)

model.compile(
    loss="sparse_categorical_crossentropy",
    optimizer=keras.optimizers.SGD(learning_rate=0.001),
    metrics=[ "sparse_categorical_accuracy" ],
)

history = model.fit(train_dataset, epochs=10, validation_data=test_dataset)
```

- Каждый сверточный и полно связанный слой, кроме выходного состоит из Convolution/Dense, также Batch Normalization и ReLU
- PointNet состоит из двух основных компонентов: основная сеть MLP (многослойный перцептрон) и трансформаторная сеть T-net.

Лекция Обработка текста

Проектирование интеллектуальных систем

Предобработка текста

Три варианта предобработки текста:

- **Стемминг.** Заключается в отбрасывании окончания
- **N-граммы.** Разделение текста на последовательности по n-слов (word2vec), либо на n-символов (вместо морфологии).
- **Морфологический анализ.** Нахождение начальной формы слова (леммы) и грамматических категорий (число, род, падеж и тд)

N-граммы



- N-граммы слов

Character n-grams

- Building a good stemmer is hard
- Cheap alternative:
 - take every n-character substring of the word
 - related words → many of the same n-grams
 - n=4,5 works well for European languages

document will describe marketing strategies by ...

docu ocum cume umen ment will desc escr scri crib rive ...

description: desc escr scri crib rive ipse prie tian
prescribing: pres hanc escr scri crib rive ibis hing
descent: desc ante spon cent
cribbage: crib rive libba libag blige

Copyright © Victor Lavrenko, 2014

- N-граммы символов

Стеммер Портера

- Простой стеммер ищет флексивную форму в таблице поиска. Недостаток – нужно перечислить все формы в таблице, поэтому незнакомые слова не обрабатываются
- Алгоритмы усечения окончаний хранят список «правил», по которым отбрасываются окончания, чтобы найти его основу
- Алгоритм стеммера Портера опубликован в 1980 году Мартином Портером. Он по правилам отсекает окончания и суффиксы



Bag-of-words

- Для Bag-of-words составляется словарь из слов текста и указывается, какое количество раз каждое из них употребляется.
- В примере три рецензии. Необходимо подсчитать количество слов в каждой

	1 This	2 movie	3 is	4 very	5 scary	6 and	7 long	8 not	9 slow	10 spooky	11 good	Length of the review(in words)
Review 1	1	1	1	1	1	1	1	0	0	0	0	7
Review 2	1	1	2	0	0	1	1	0	1	0	0	8
Review 3	1	1	1	0	0	0	1	0	0	1	1	6

TF-IDF

Далее вычисляем метрику TF – частота употребления слова в документе

$$tf(t, d) = \frac{n_t}{\sum_k n_k}$$

Term	Review 1	Review 2	Review 3	TF (Review 1)	TF (Review 2)	TF (Review 3)
This	1	1	1	1/7	1/8	1/6
movie	1	1	1	1/7	1/8	1/6
is	1	2	1	1/7	1/4	1/6
very	1	0	0	1/7	0	0
scary	1	1	0	1/7	1/8	0
and	1	1	1	1/7	1/8	1/6
long	1	0	0	1/7	0	0
not	0	1	0	0	1/8	0
slow	0	1	0	0	1/8	0
spooky	0	0	1	0	0	1/6
good	0	0	1	0	0	1/6

где n_t есть число вхождений слова t в документ, а в знаменателе — общее число слов в данном документе.

TF-IDF

Далее вычисляется метрика IDF, и ее значение умножается на TF

$$\text{idf}(t, D) = \log \frac{|D|}{|\{d_i \in D \mid t \in d_i\}|}$$

Term	Review 1	Review 2	Review 3	IDF	TF-IDF (Review 1)	TF-IDF (Review 2)	TF-IDF (Review 3)
This	1	1	1	0.00	0.000	0.000	0.000
movie	1	1	1	0.00	0.000	0.000	0.000
is	1	2	1	0.00	0.000	0.000	0.000
very	1	0	0	0.48	0.068	0.000	0.000
scary	1	1	0	0.18	0.025	0.022	0.000
and	1	1	1	0.00	0.000	0.000	0.000
long	1	0	0	0.48	0.068	0.000	0.000
not	0	1	0	0.48	0.000	0.060	0.000
slow	0	1	0	0.48	0.000	0.060	0.000
spooky	0	0	1	0.48	0.000	0.000	0.080
good	0	0	1	0.48	0.000	0.000	0.080

- IDI — число документов в коллекции;
- $|\{d_i \in D \mid t \in d_i\}|$ — число документов из коллекции D , в которых встречается t (когда $n_t \neq 0$).

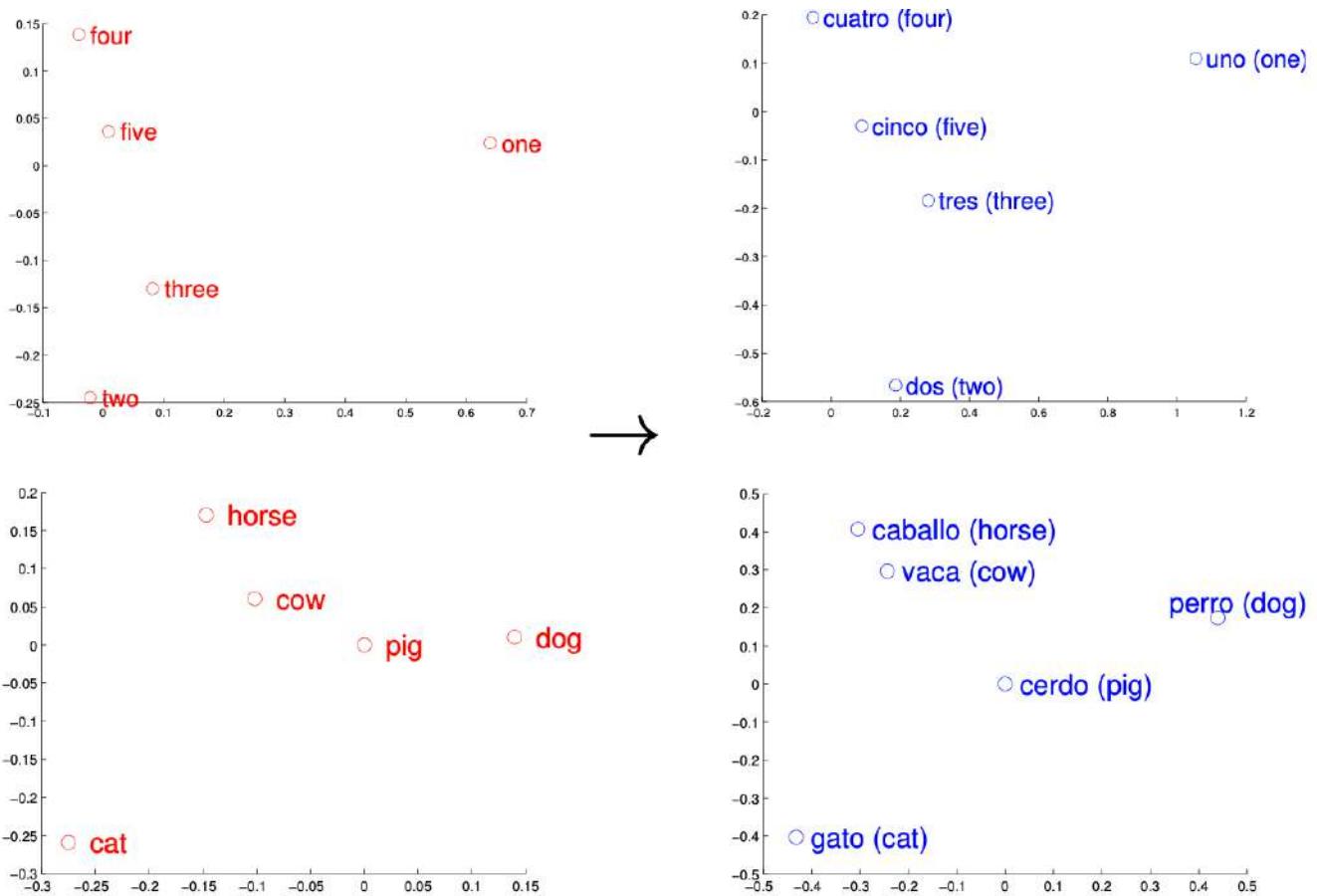
$$\text{tf-idf}(t, d, D) = \text{tf}(t, d) \times \text{idf}(t, D)$$

Word embedding

- Классическое представление bag-of-words плохо подходит для представления данных для обучения нейронных сетей.
- Размерность такого пространства признаков оказывается очень большой, равной количеству слов в словаре.
- Поэтому оказывается очень полезным использовать векторное представление слов (embedding). Помимо сокращения пространства признаков это позволяет близкие к друг другу слова располагать ближе в этом пространстве.

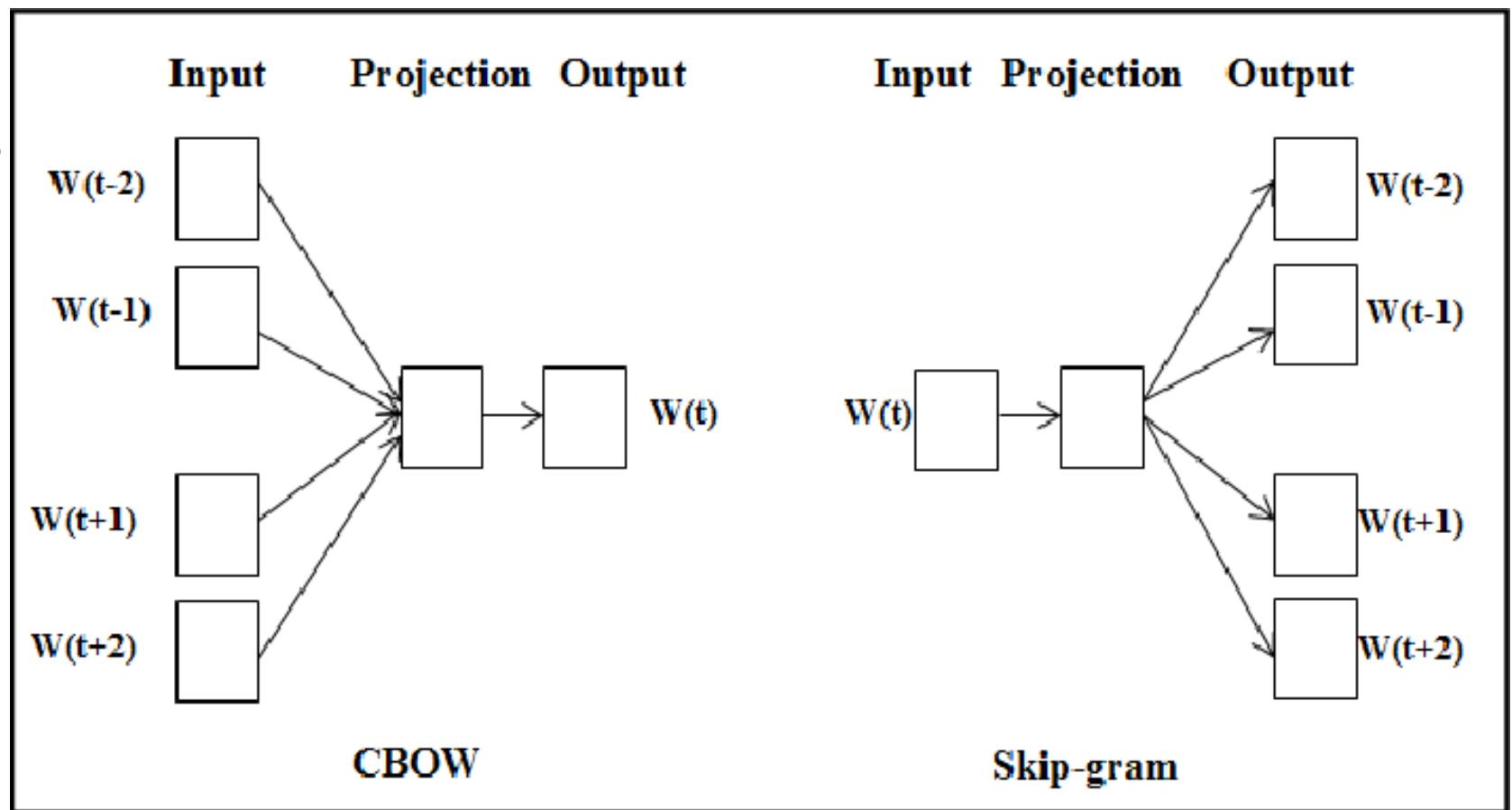
Word2vec

- Разработано в Google в 2013 году. По большому корпусу данных вычисляется векторное представление слов (embedding), обучаясь на этих данных.
- Каждому слову соответствует вектор в этом пространстве. Схожие по смыслу слова находятся в этом пространстве рядом.
- Используется модель из одного скрытого слоя.
- Между представлениями для разных языков также наблюдается зависимость



CBOW и Skip-gram

- Используются два вида представления: CBOW (Continuous Bag of Words) и Skip-Gram. Эти два представления оперируют с определенным окном входных данных.
- CBOW предсказывает слово исходя из контекста.
- Skip-Gram предлагает список вероятного контекста в рамках окна для выбранного слова.
- В обоих случаях порядок слов не анализируется.



From Words to Numbers

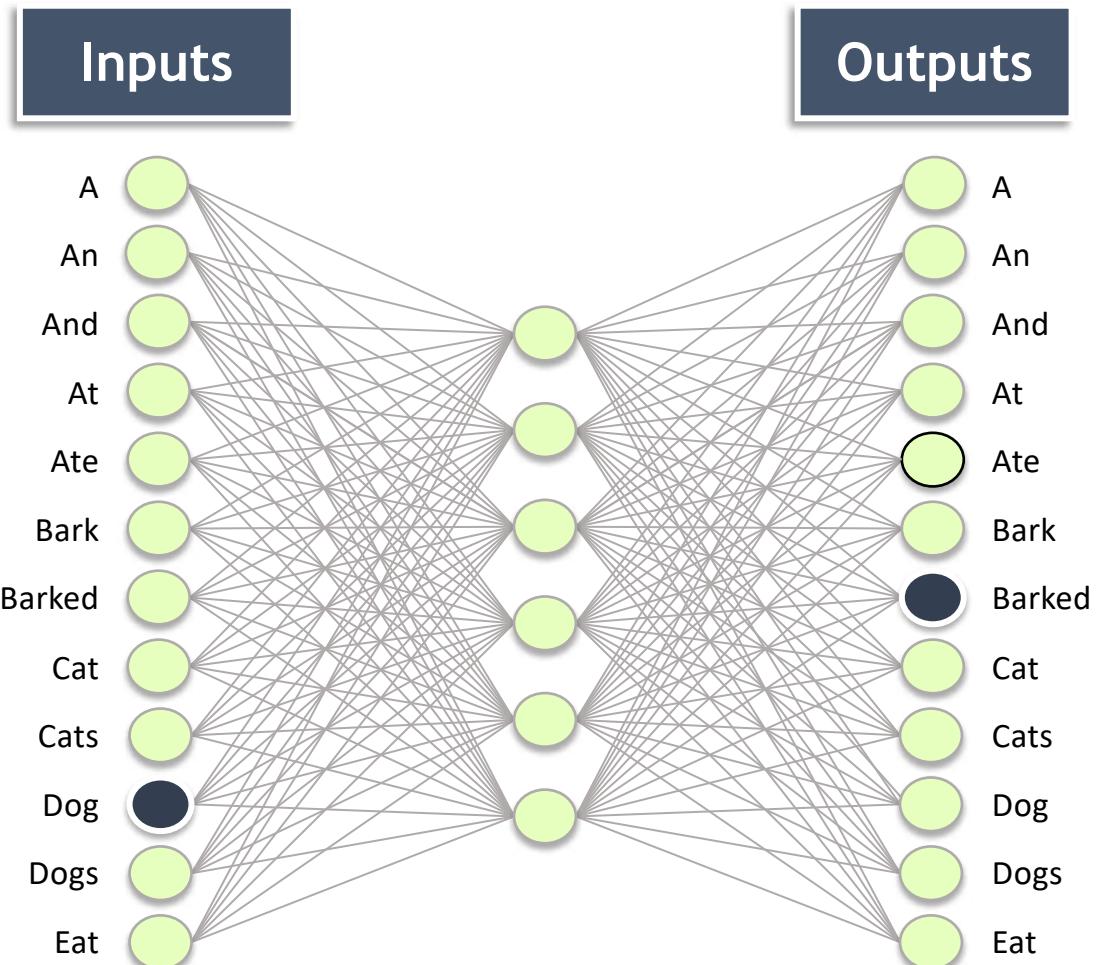
“A dog barked at a cat.”

[1, 10, 7, 4, 1, 8]

DICTIONARY

1. A	8. CAT
2. AN	9. CATS
3. AND	10. DOG
4. AT	11. DOGS
5. ATE	12. EAT
6. BARK	
7. BARKED	

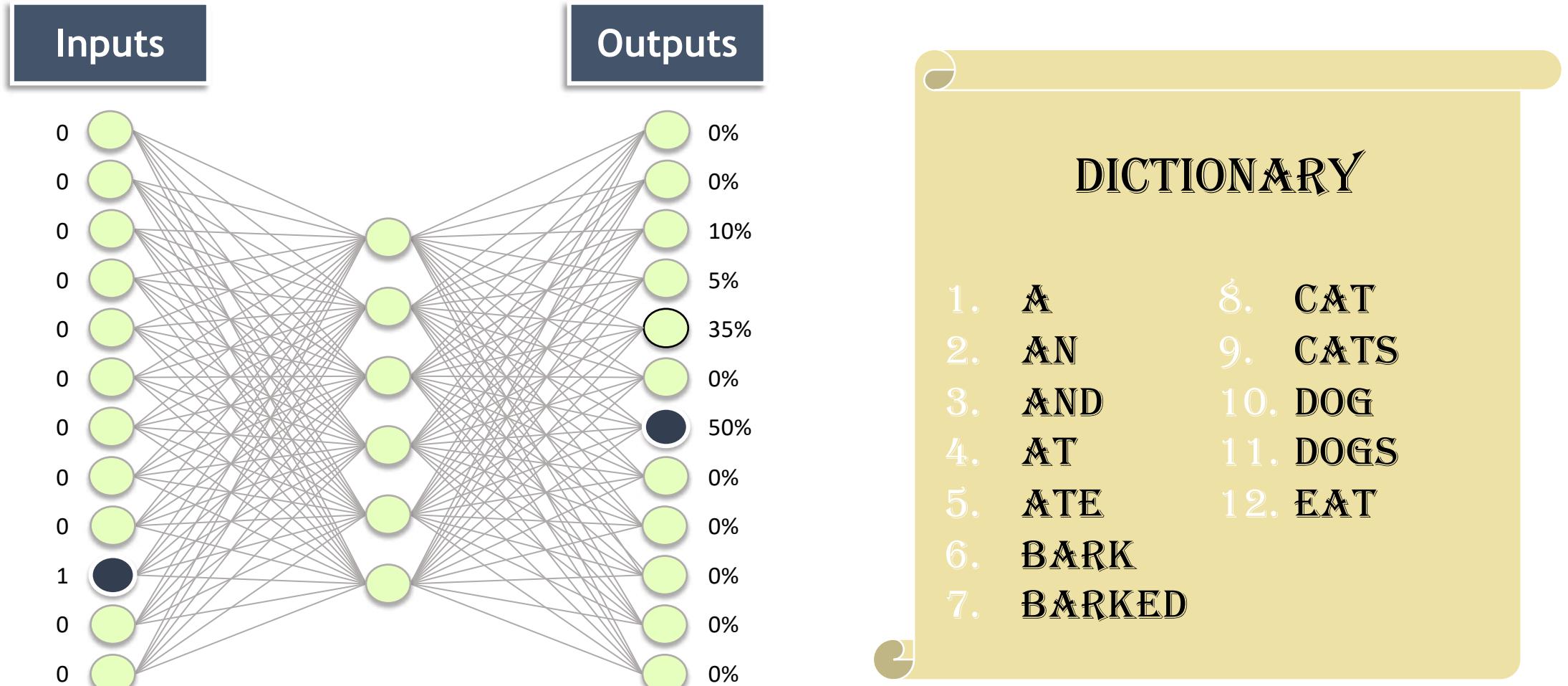
From Words to Numbers



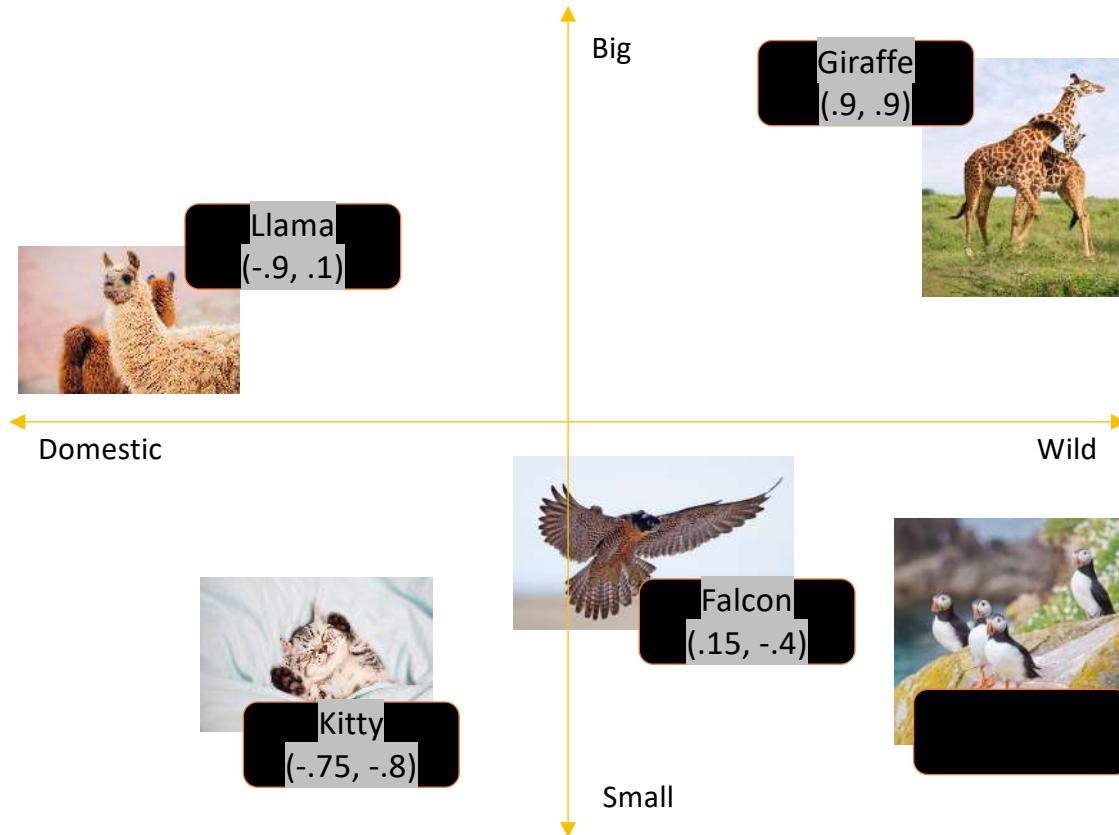
DICTIONARY

- | | |
|------------------|-----------------|
| 1. A | 8. CAT |
| 2. AN | 9. CATS |
| 3. AND | 10. DOG |
| 4. AT | 11. DOGS |
| 5. ATE | 12. EAT |
| 6. BARK | |
| 7. BARKED | |

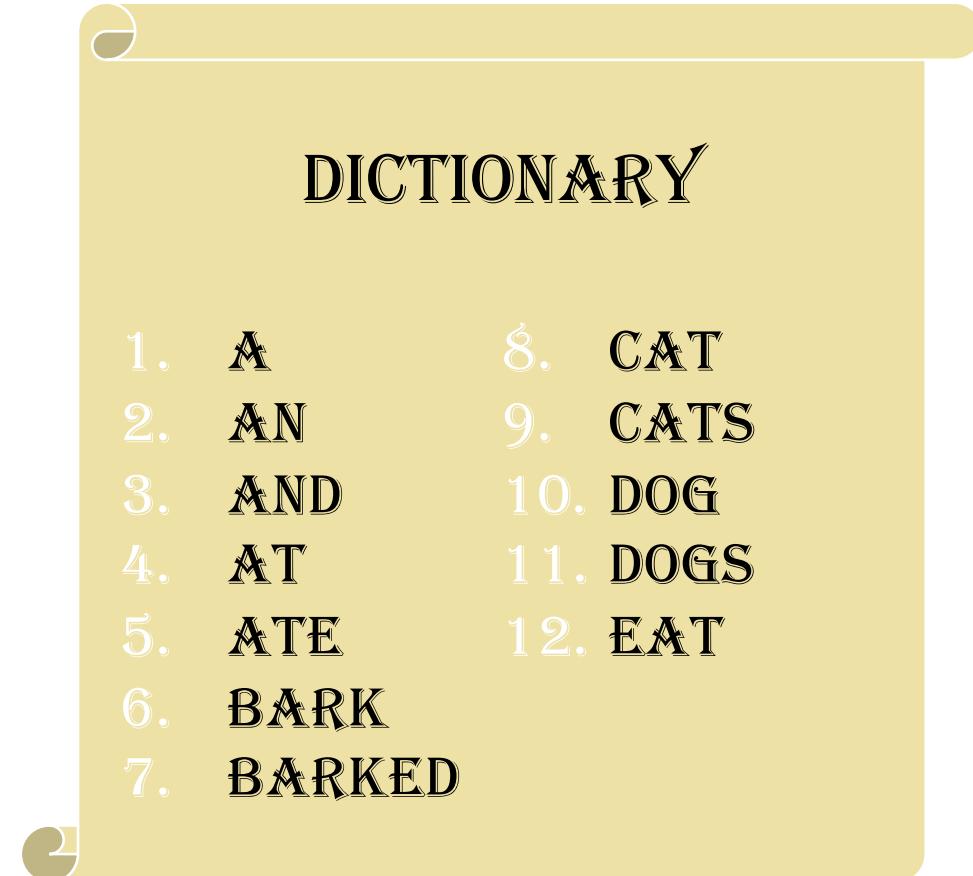
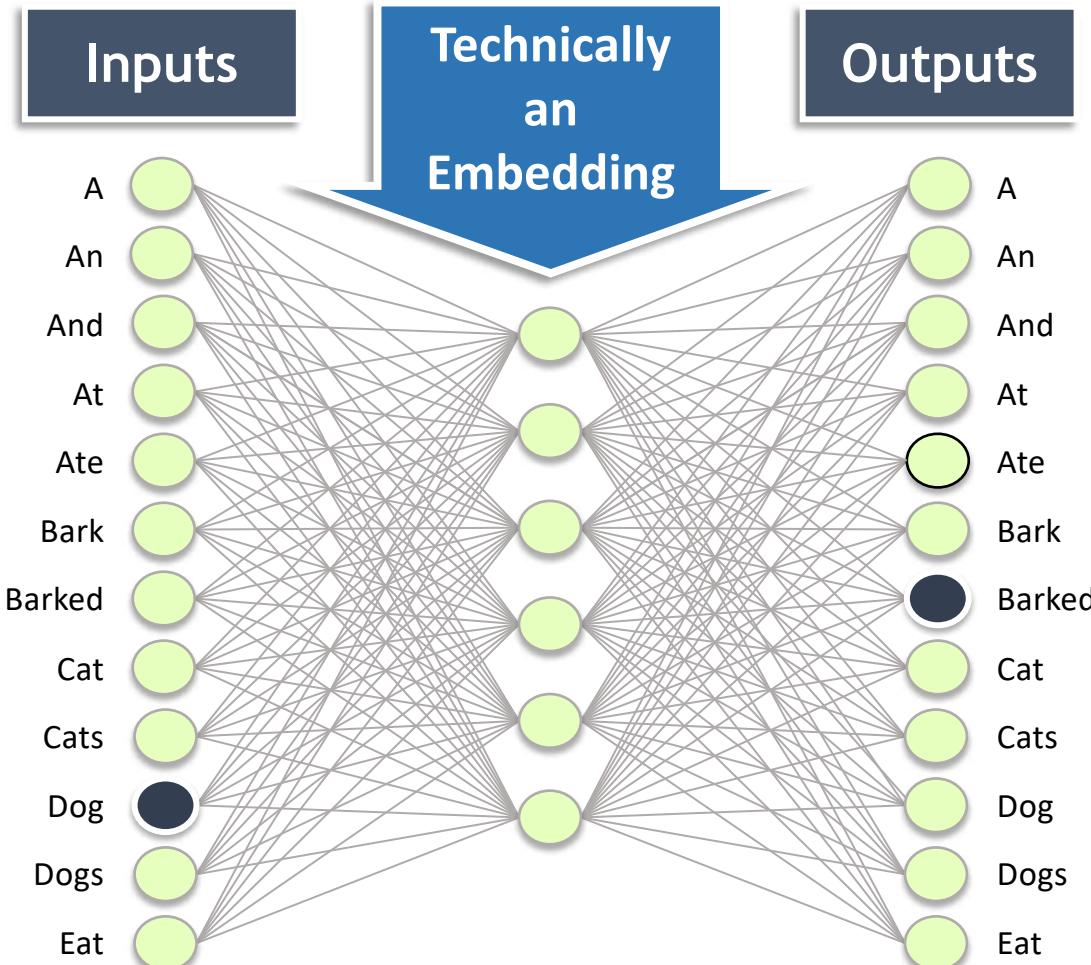
From Words to Numbers



From Words to Numbers

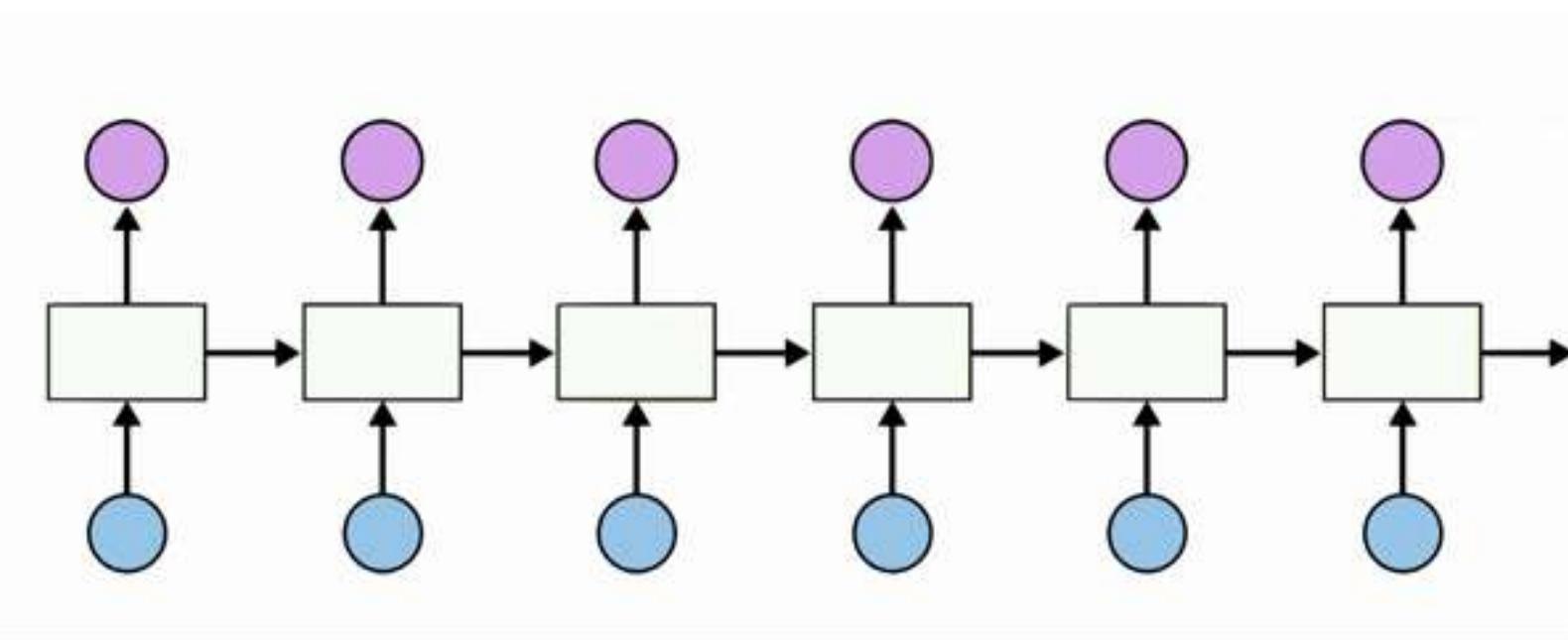


From Words to Numbers



RNN

- Рекуррентная нейронная сеть учитывает
состояние ячейки



Recurrent Neural Networks

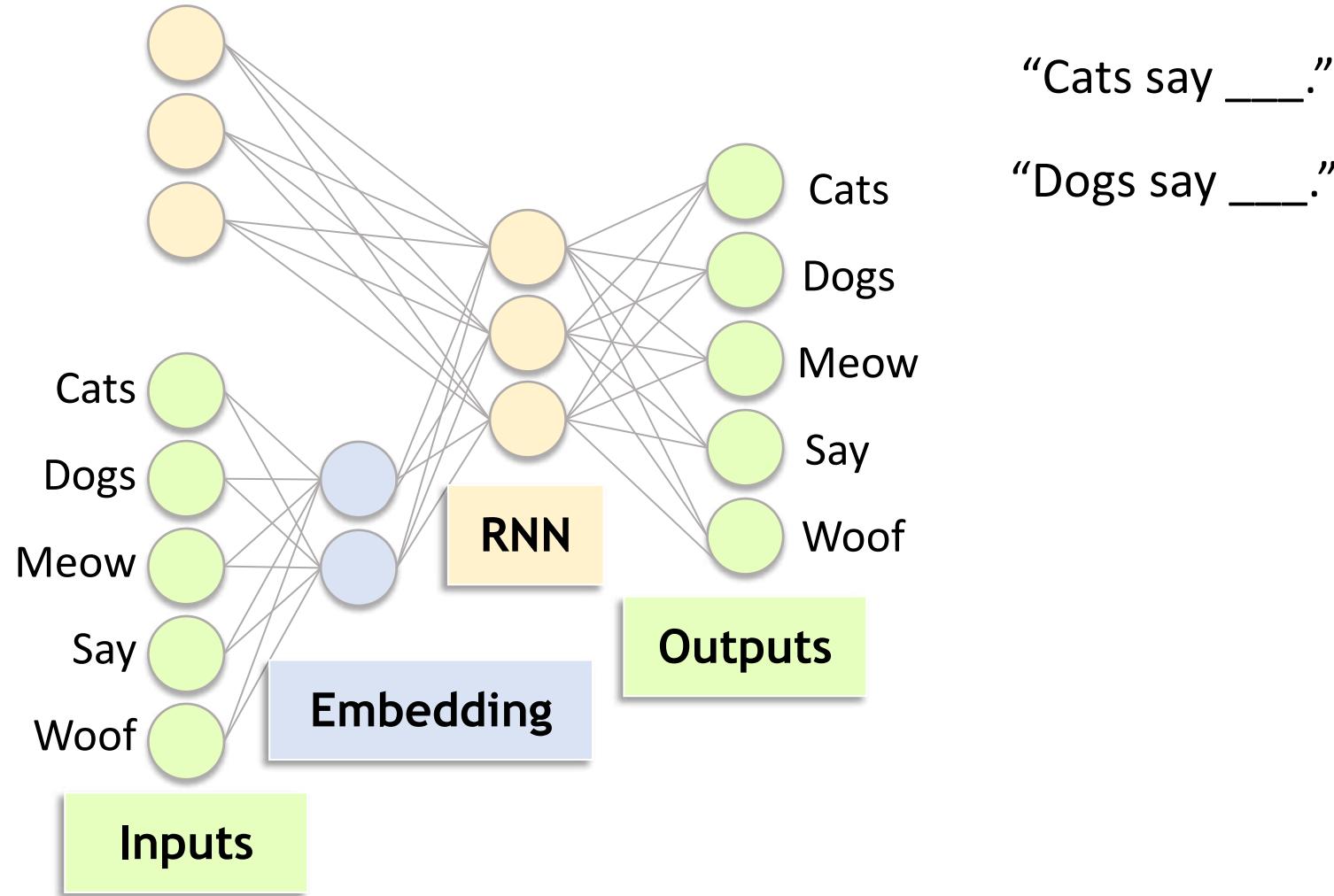
“Cats say ____.”

“Dogs say ____.”

DICTIONARY

1. CATS
2. DOGS
3. MEOW
4. SAY
5. WOOF

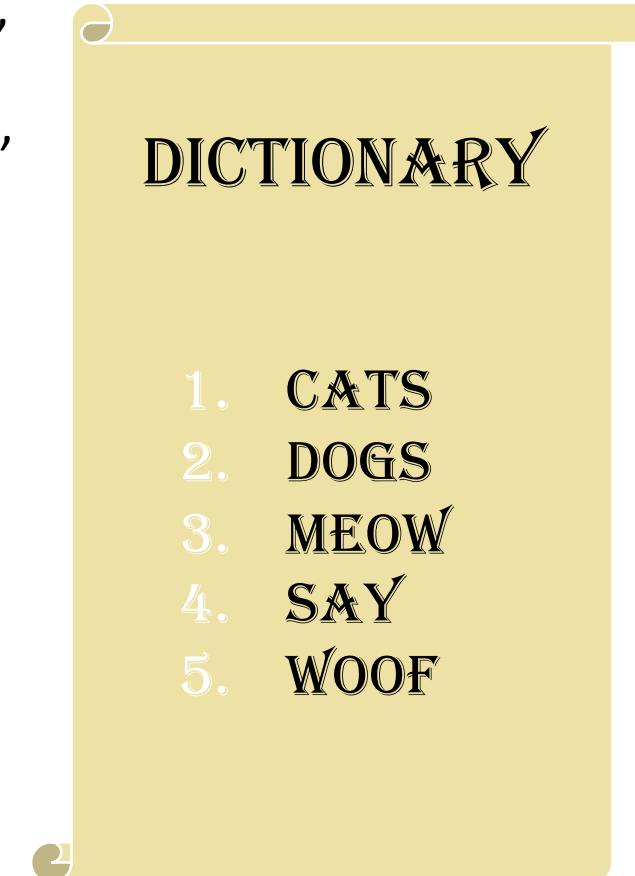
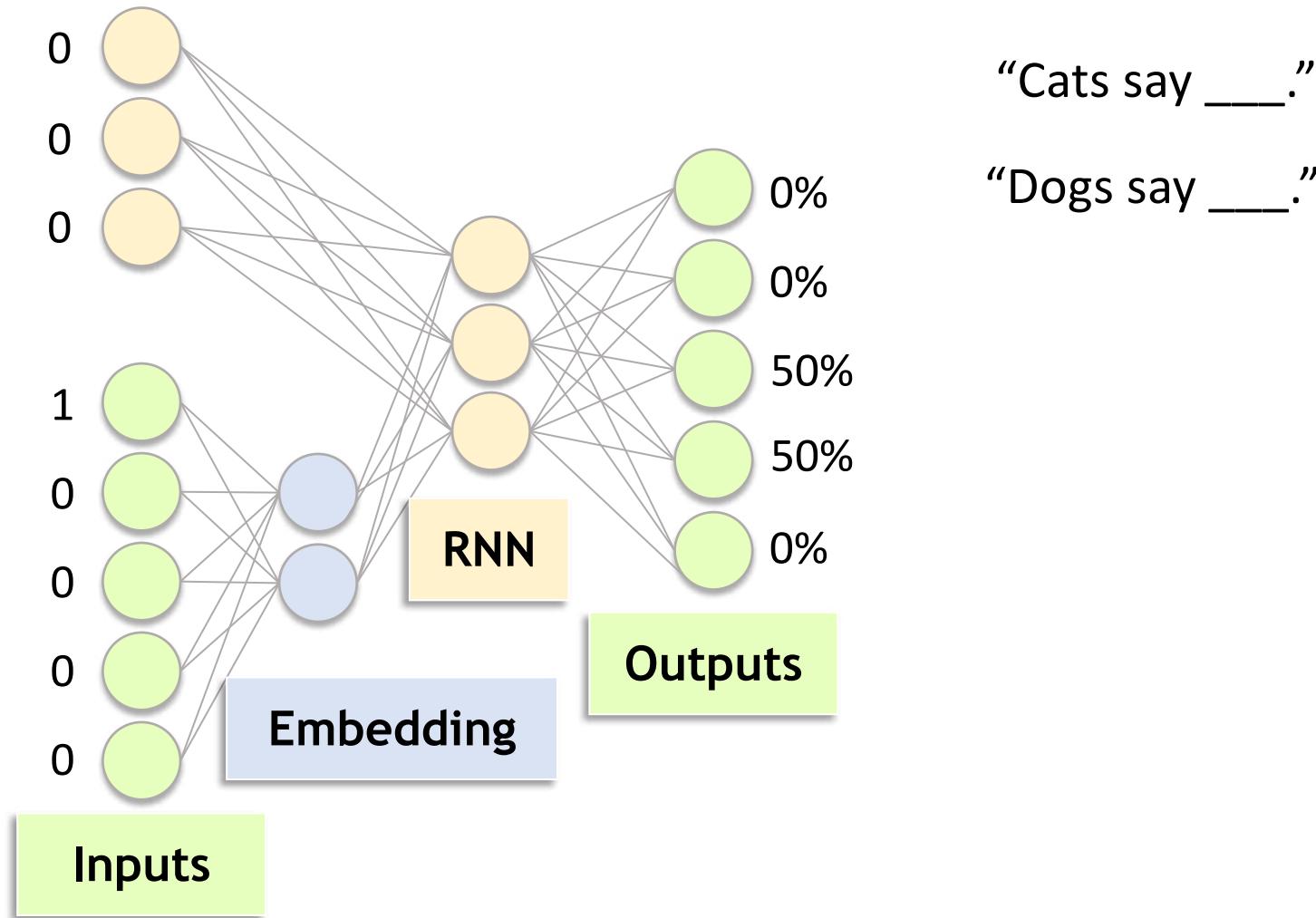
Recurrent Neural Networks



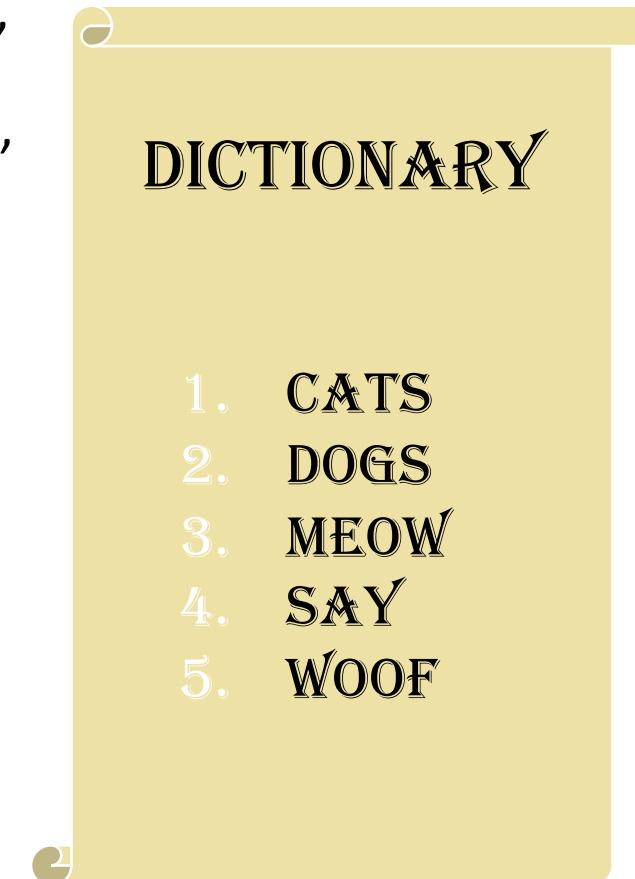
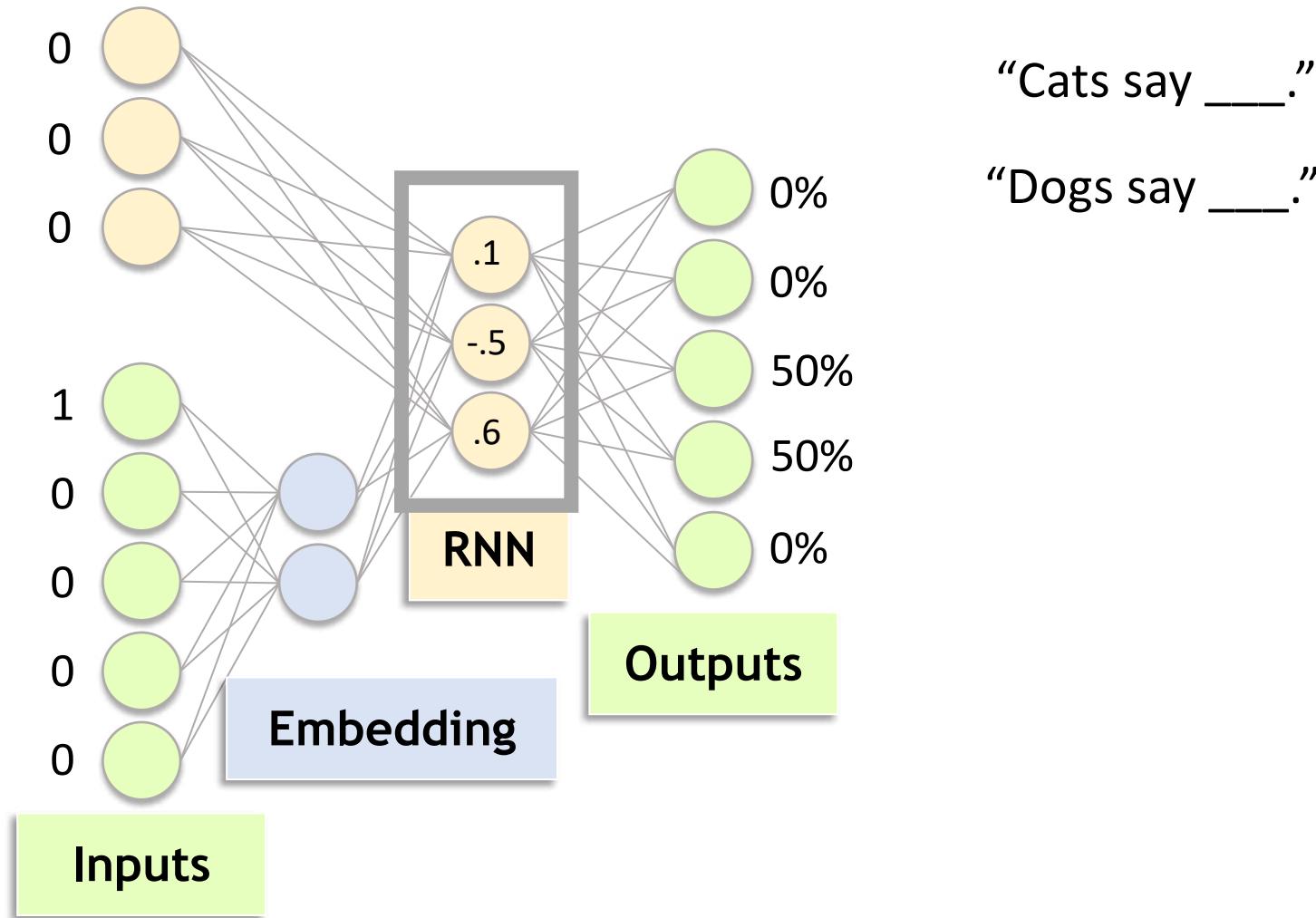
DICTIONARY

1. CATS
2. DOGS
3. MEOW
4. SAY
5. WOOF

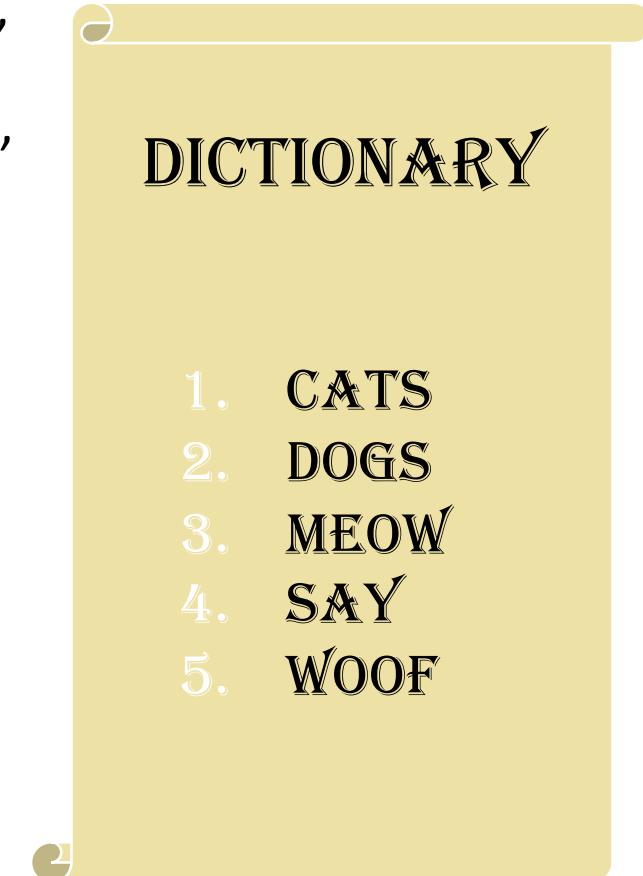
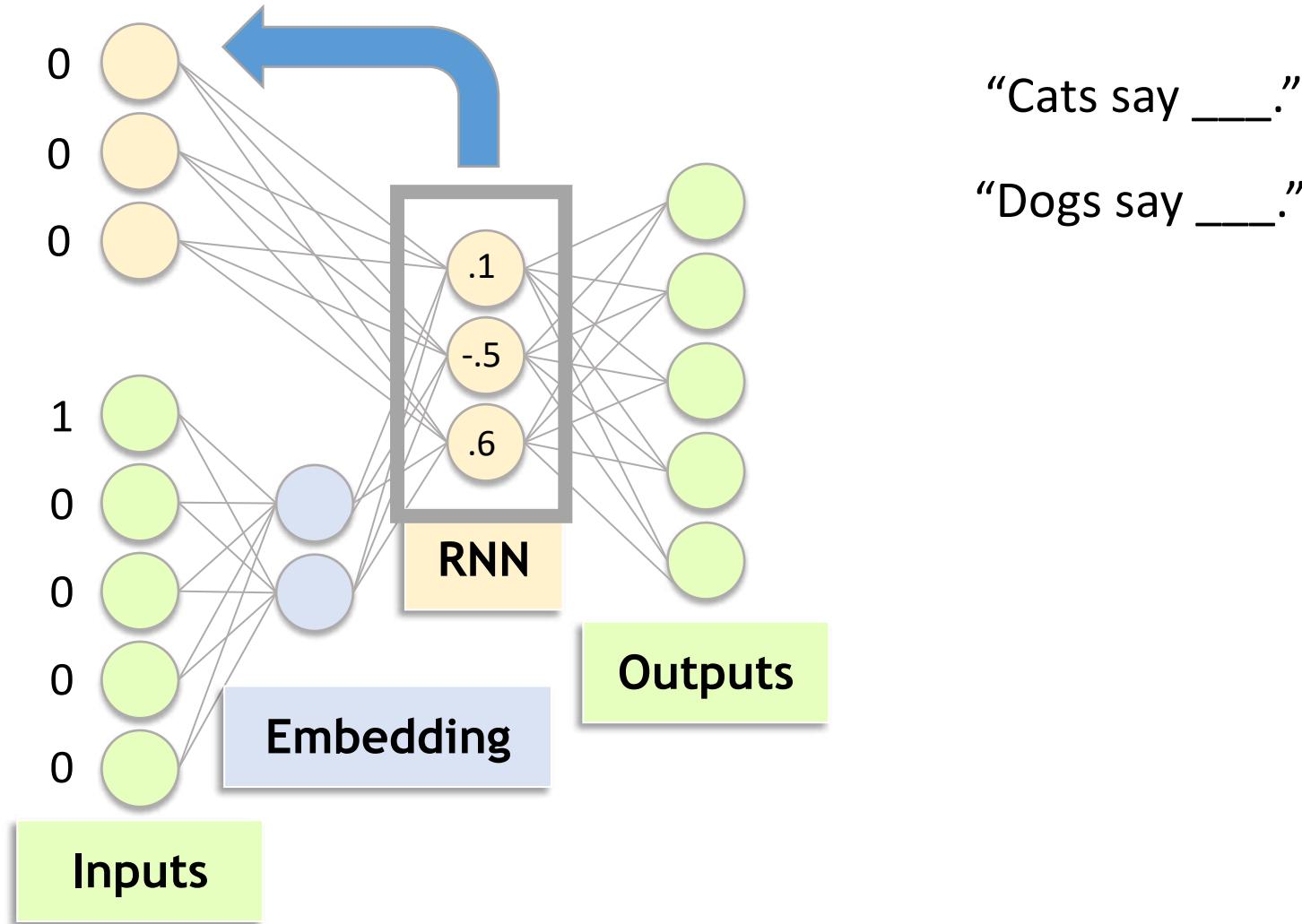
Recurrent Neural Networks



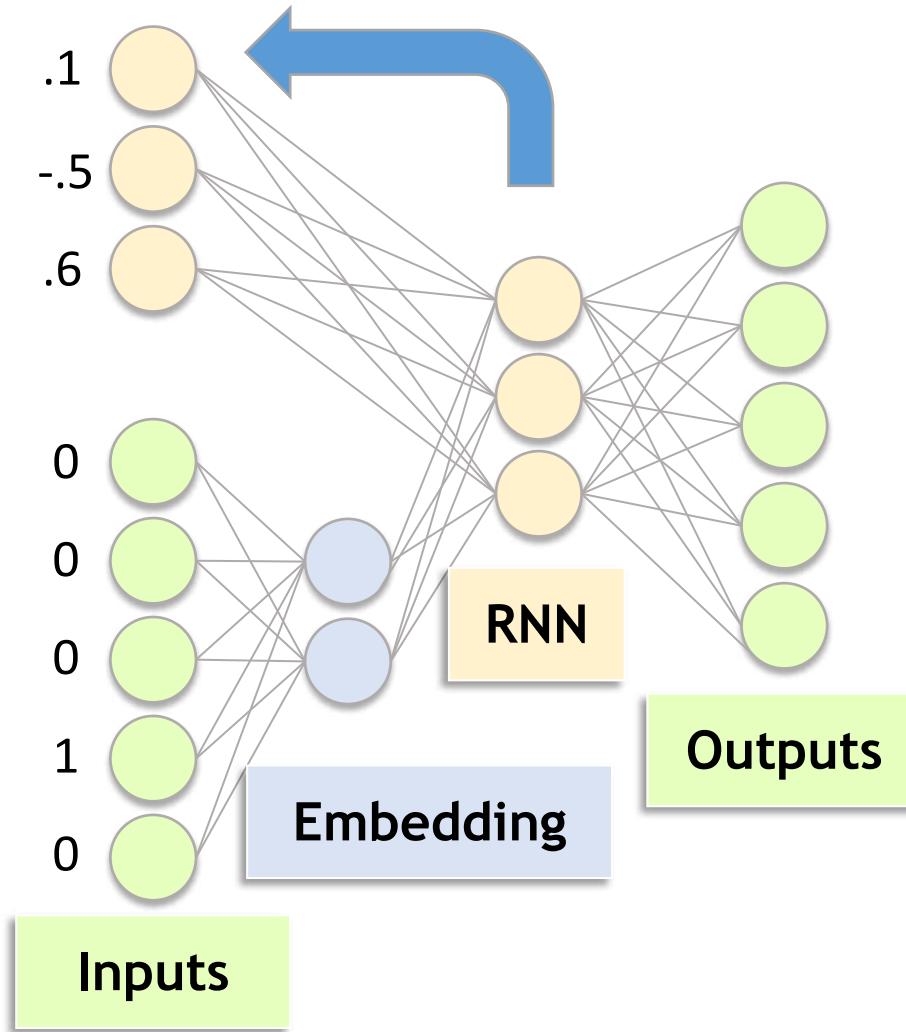
Recurrent Neural Networks



Recurrent Neural Networks



Recurrent Neural Networks



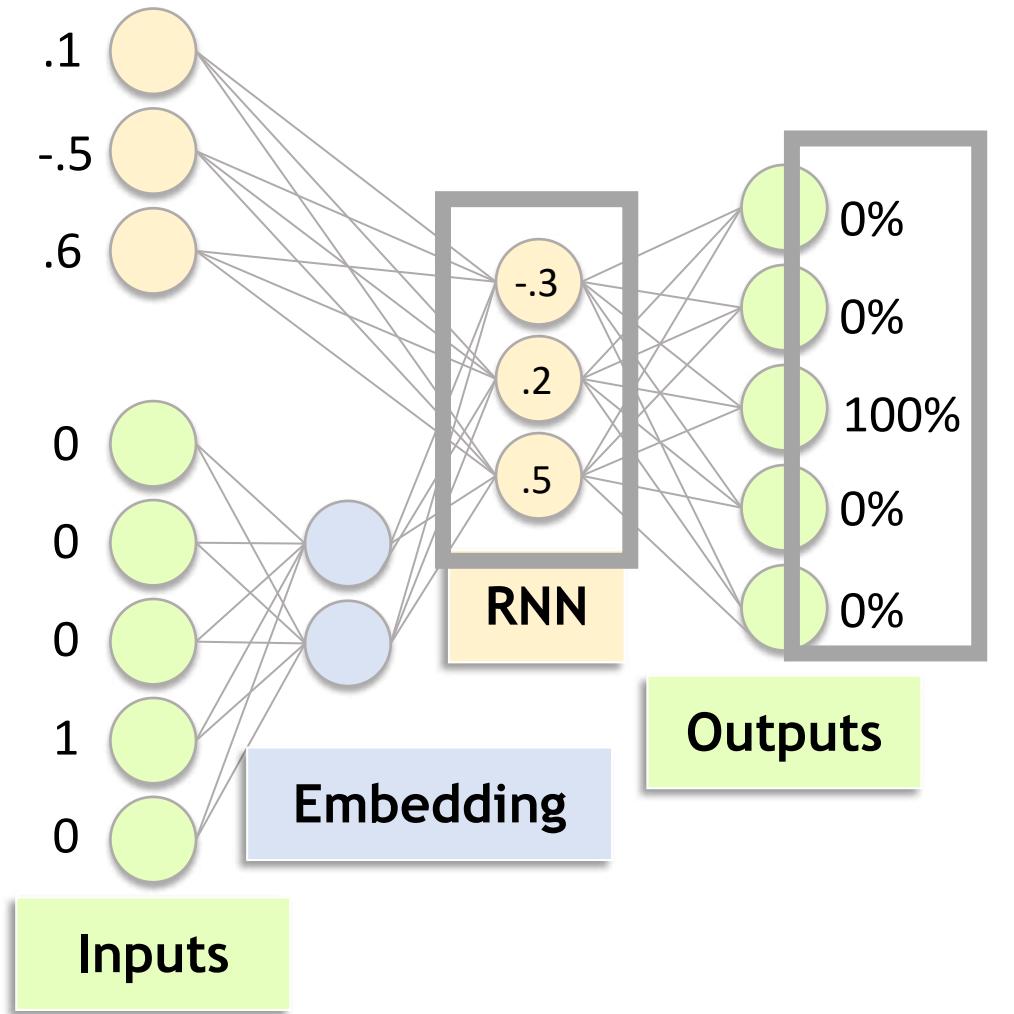
“Cats say ____.”

“Dogs say ____.”

DICTIONARY

1. CATS
2. DOGS
3. MEOW
4. SAY
5. WOOF

Recurrent Neural Networks



“Cats say ____.”

“Dogs say ____.”

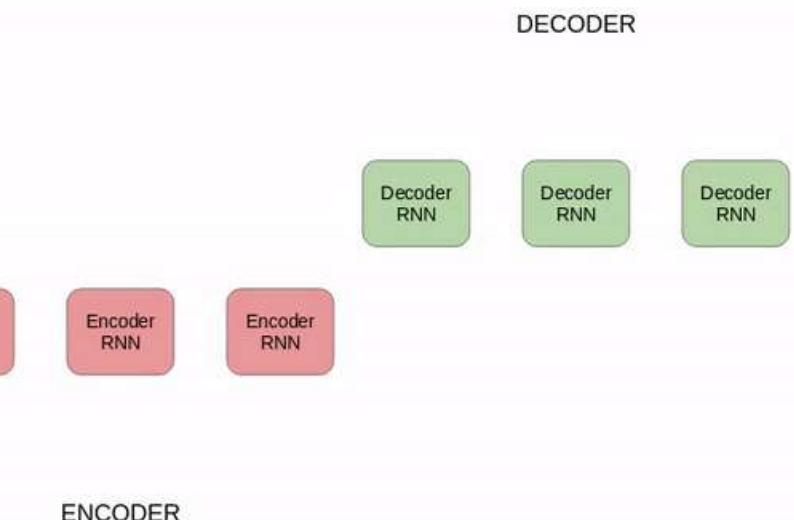
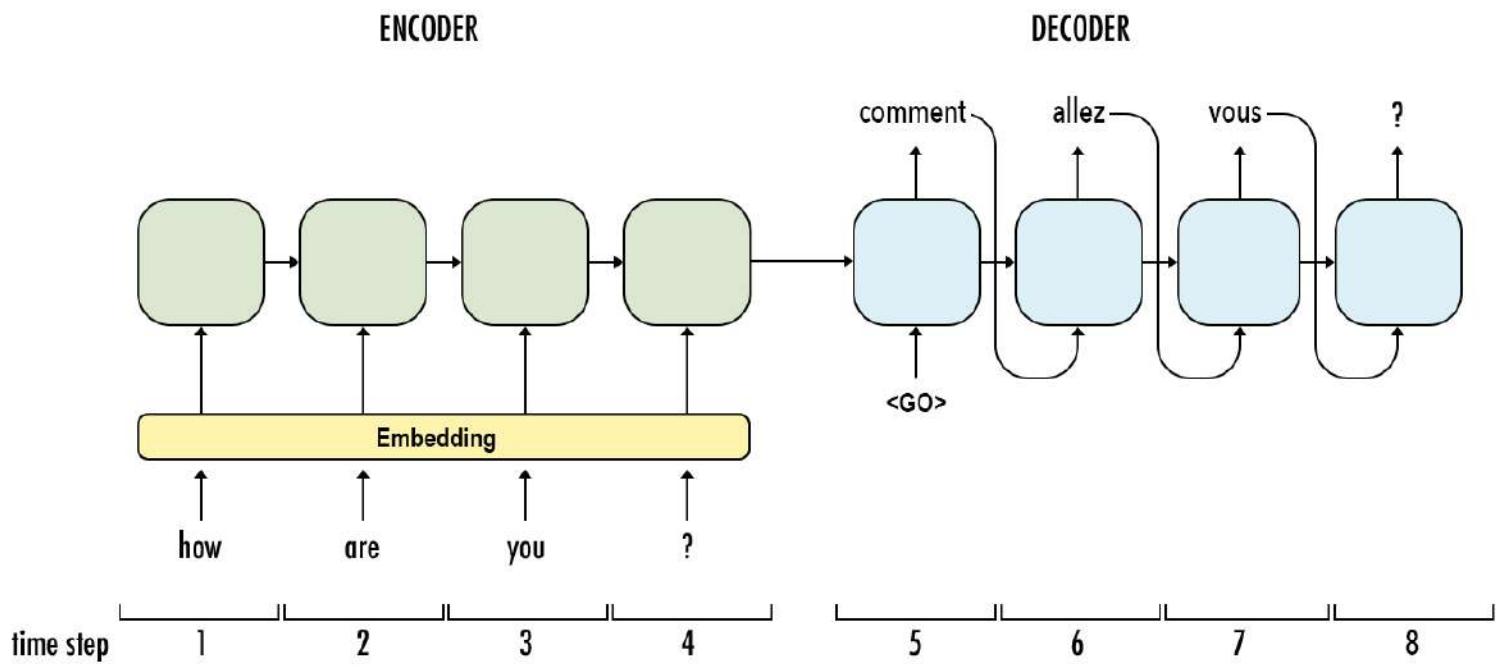
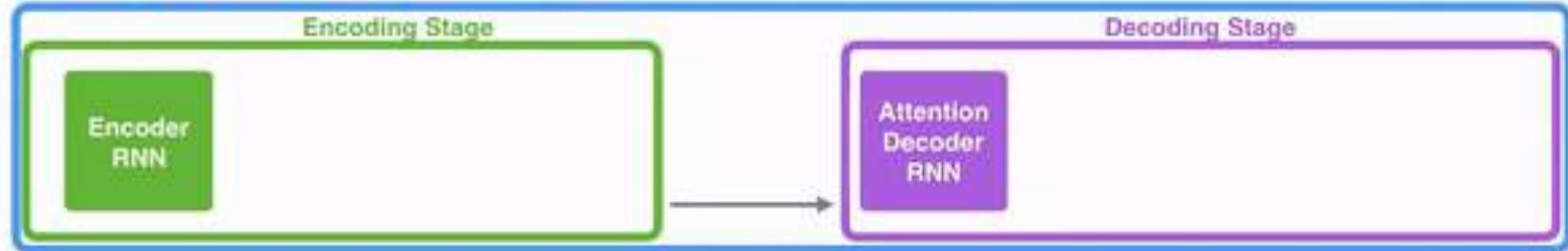
DICTIONARY

1. CATS
2. DOGS
3. MEOW
4. SAY
5. WOOF

Seq2seq

Для машинного перевода

Neural Machine Translation SEQUENCE TO SEQUENCE MODEL WITH ATTENTION



Лекция 7

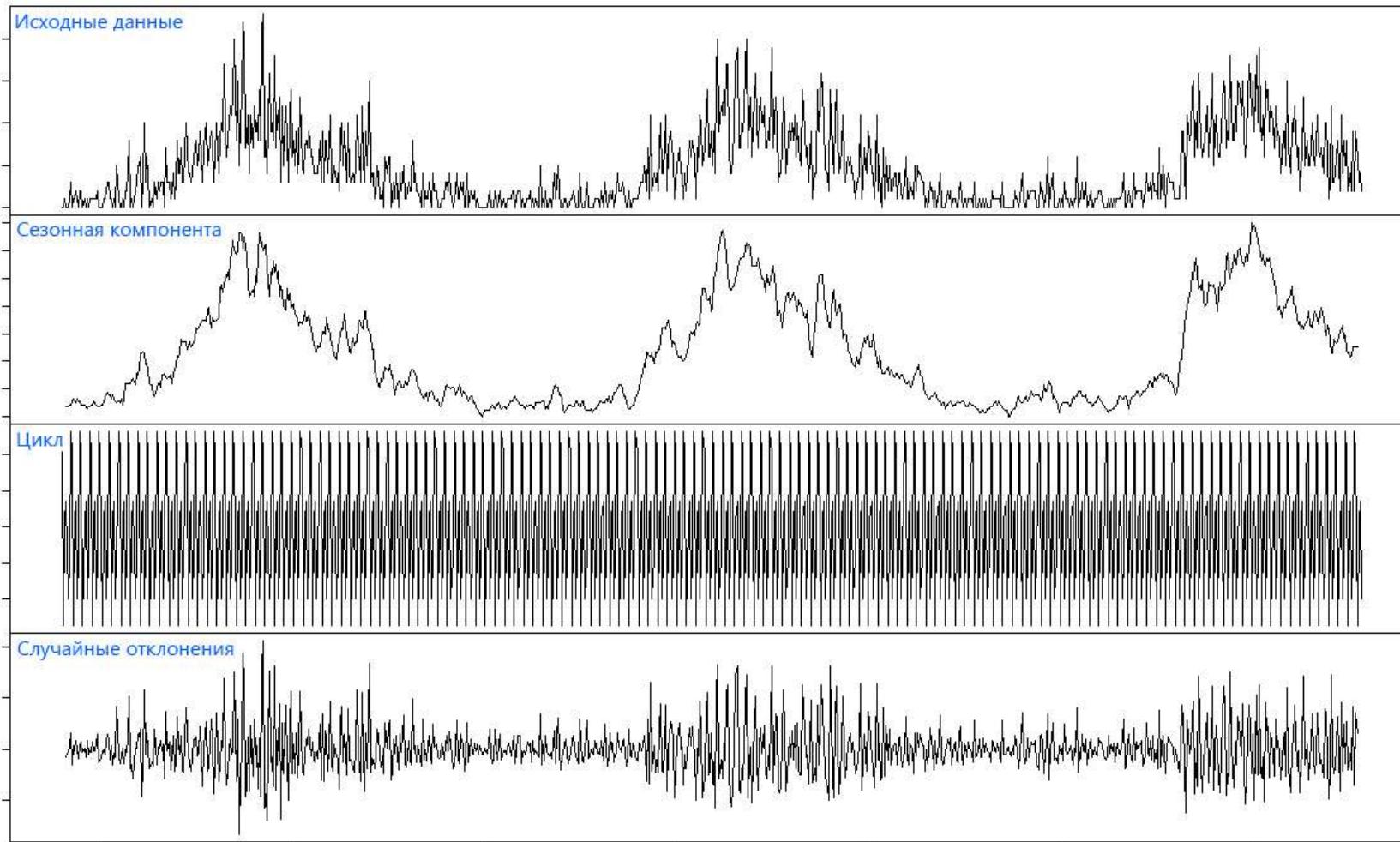
Временные ряды и рекуррентные нейронные сети

Разработка нейросетевых систем

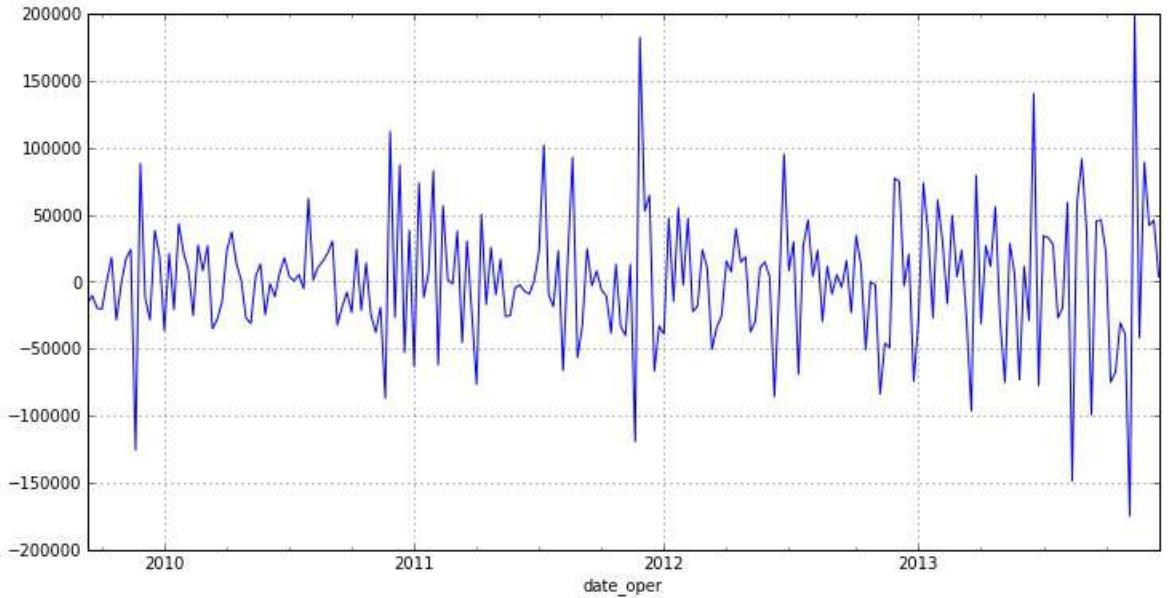
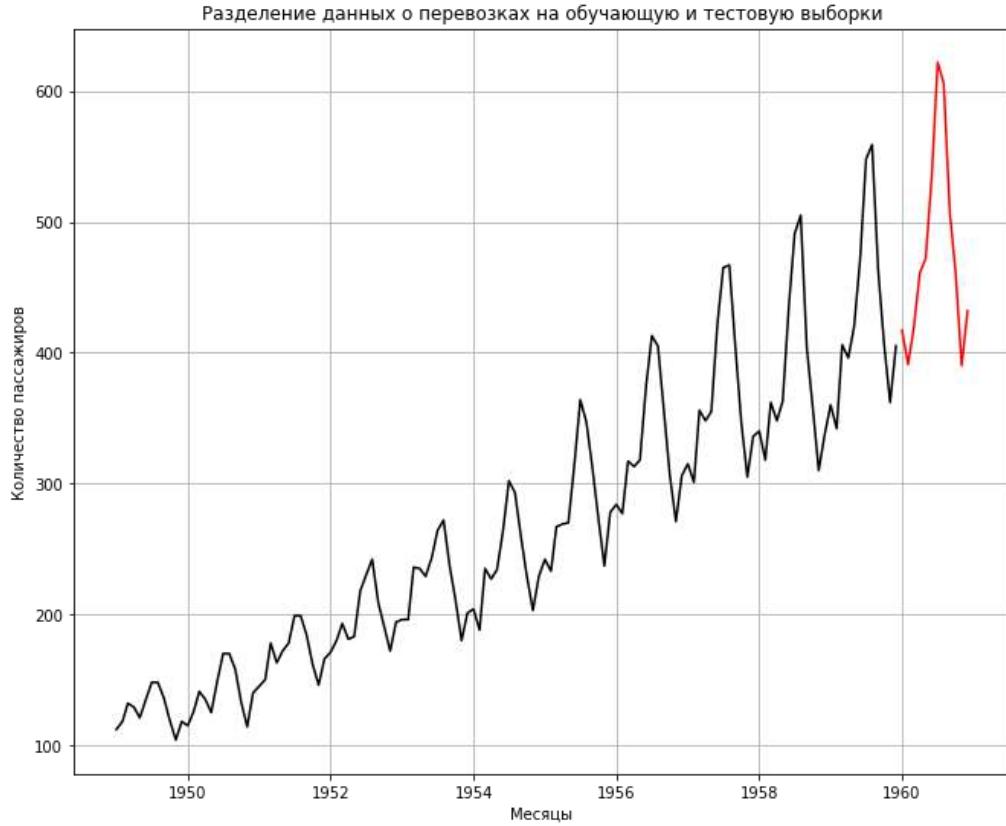
Временной ряд

- **Временной ряд** — собранный в разные моменты времени статистический материал о значении каких-либо параметров.
- Временной ряд существенно отличается от простой выборки данных, так как при анализе учитывается взаимосвязь измерений со временем, а не только статистическое разнообразие и статистические характеристики выборки.
- Стационарный временной ряд – ряд, у которого не меняются математическое ожидание и дисперсия со временем.

Преобразование временных рядов



Стационарность временного ряда



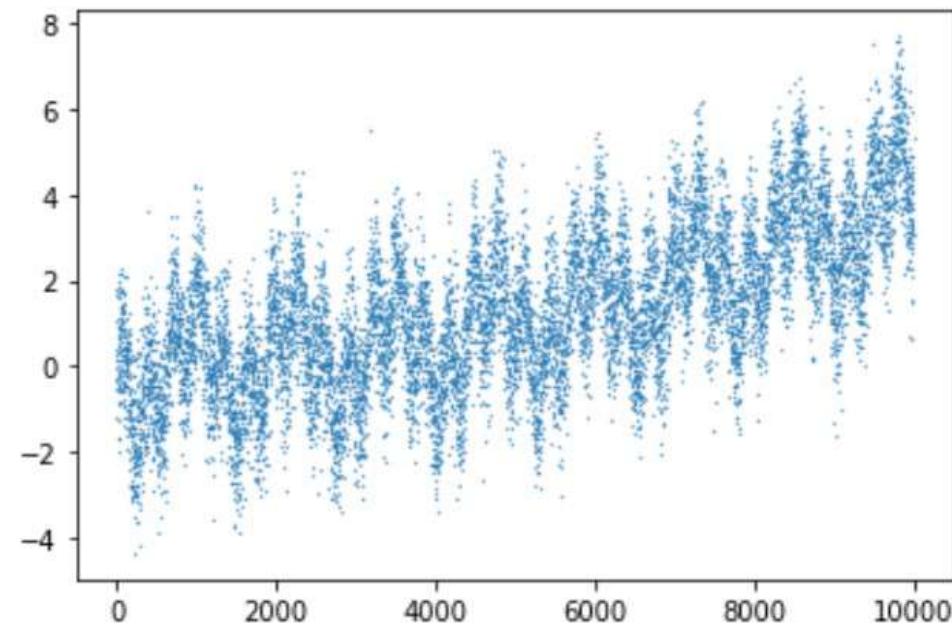
- Стационарный ряд имеет постоянный тренд и только случайная компонента
- В нестационарном может меняться тренд или присутствует циклическая компонента

Синтетические данные

```
X = np.arange(10000)
y = np.sin(X/50)-np.sin(X/200)+(2*X/X.size)**2
y += np.random.normal(scale=1.0, size=y.size)
plt.scatter(X, y[:10000], s=0.1)

df = pd.Series(y)
df = df.diff().dropna()
```

- Первая часть лабораторной посвящена анализу синтетических данных
- Создадим ряд со всеми тремя компонентами

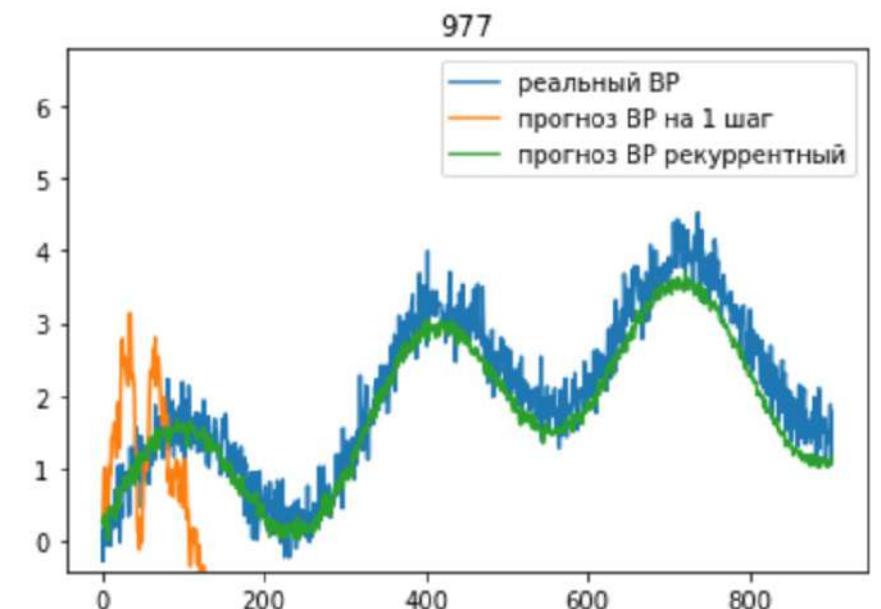
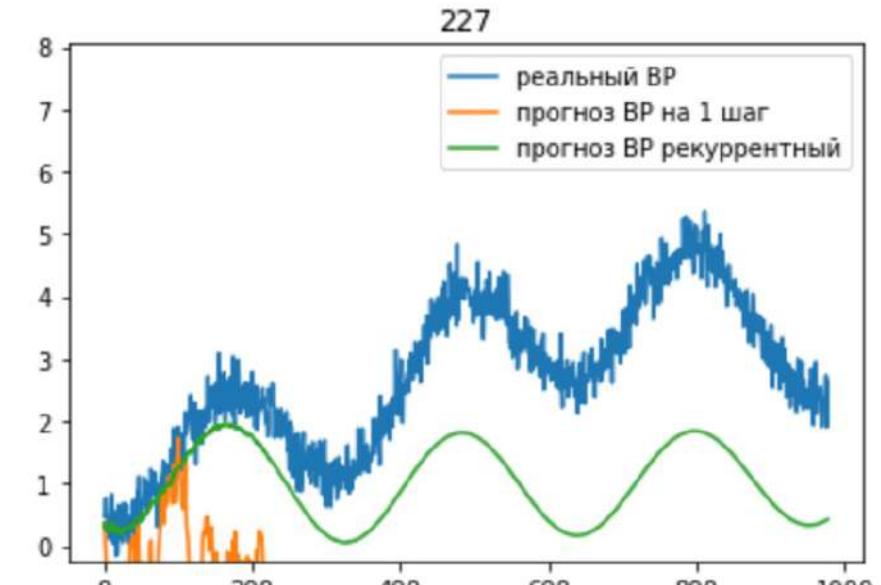


Модель авторегрессии

- **Авторегрессионная (AR) модель** (autoregressive model) — модель временных рядов, в которой значения временного ряда в данный момент линейно зависят от предыдущих значений этого же ряда.
- Авторегрессионный процесс порядка p (AR(p)-процесс) определяется следующим образом

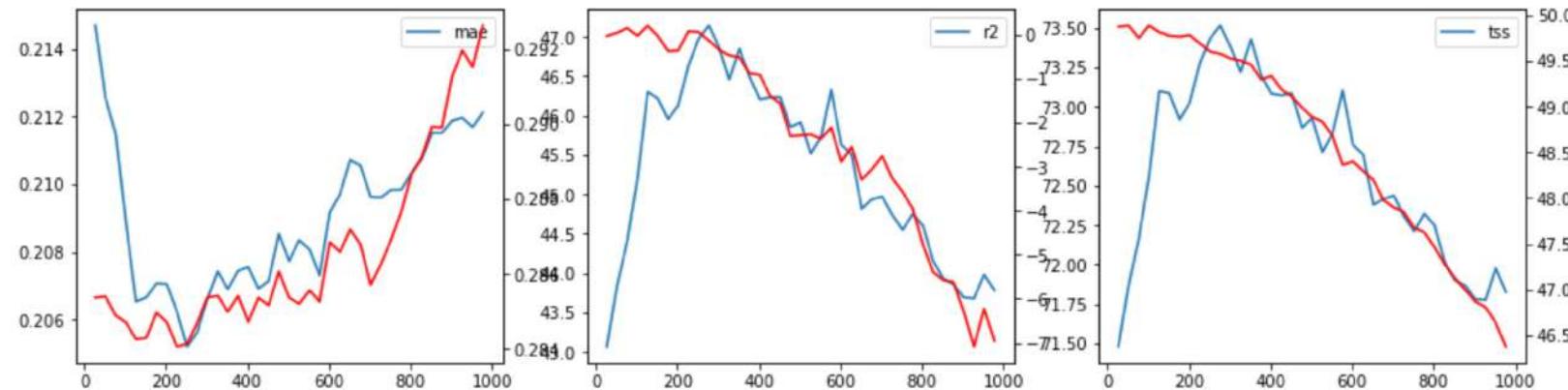
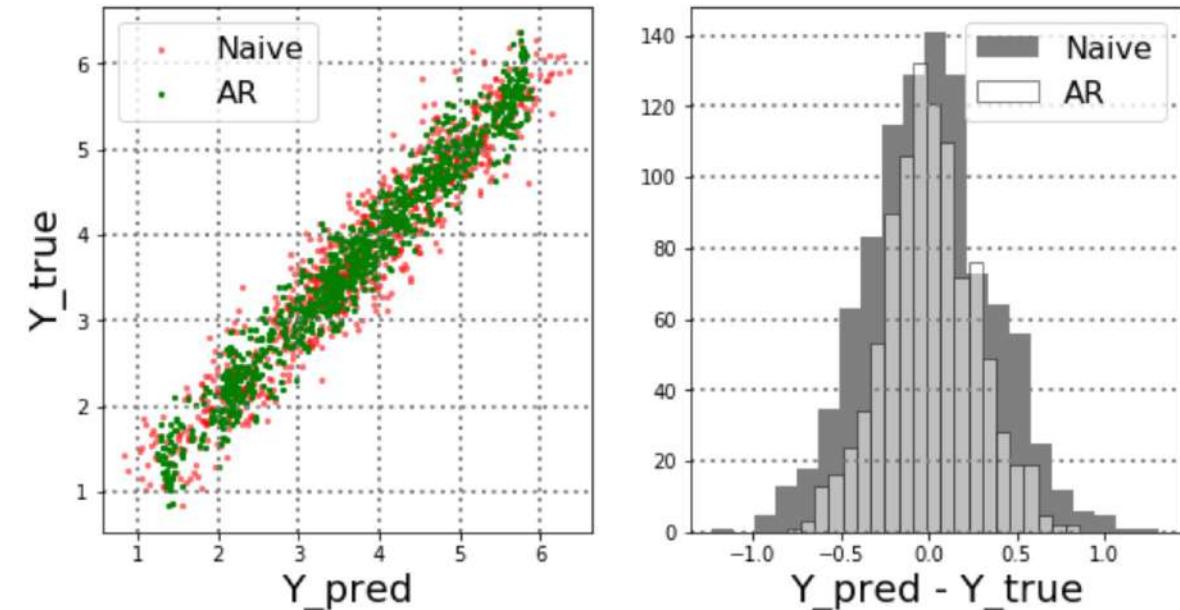
$$X_t = c + \sum_{i=1}^p a_i X_{t-i} + \varepsilon_t.$$

- a - **параметры** модели (коэффициенты авторегрессии)
- c - **постоянная** (часто для упрощения предполагается равной нулю)
- ε - **белый шум**



Результаты

- Выведем график плотности распределения **разности** предсказанного и истинного значения



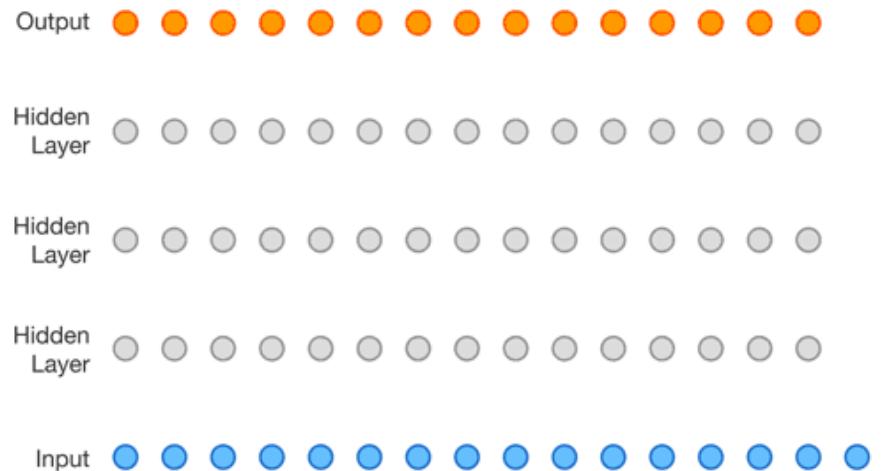
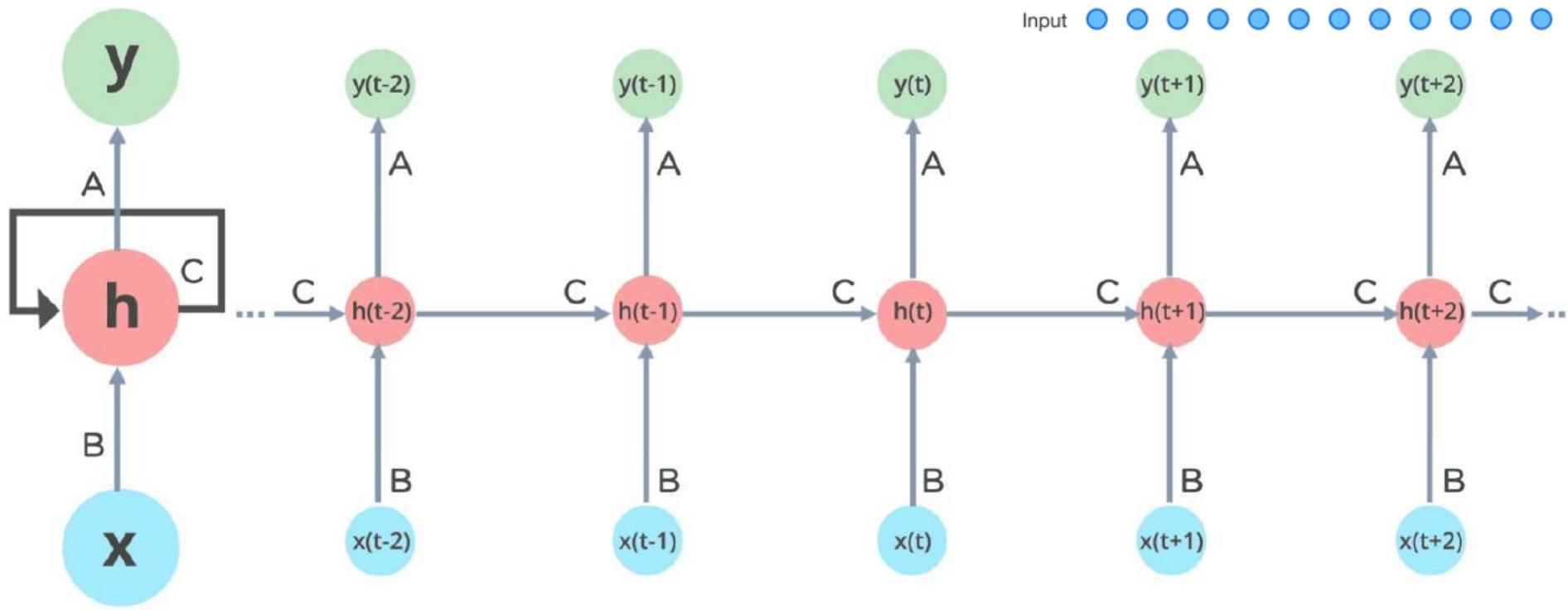
- Выведем графики средней абсолютной ошибки и коэффициента детерминации

Рекуррентная нейронная сеть

- **Рекуррентные нейронные сети** (*Recurrent neural network; RNN*) — вид нейронных сетей, где связи между элементами образуют направленную последовательность.
- Благодаря этому появляется возможность обрабатывать серии событий во времени или последовательные пространственные цепочки.
- В отличие от многослойных перцепtronов, рекуррентные сети могут использовать свою внутреннюю память для обработки последовательностей произвольной длины.

Рекуррентная нейросеть

- Рекуррентная нейросеть позволяет реализовать память
- GRU, LSTM



Рекуррентная нейронная сеть

- Сеть Элмана

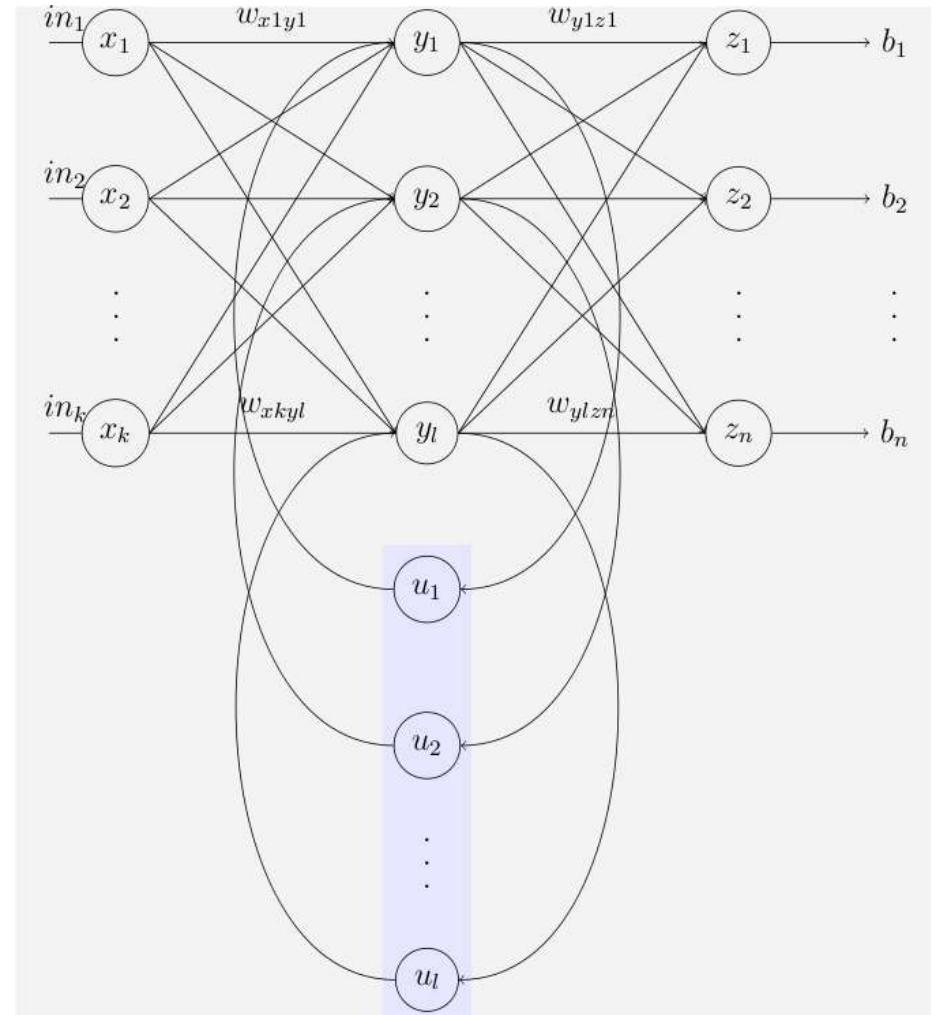
$$h_t = \sigma_h(W_h x_t + U_h h_{t-1} + b_h)$$

$$y_t = \sigma_y(W_y h_t + b_y)$$

- Сеть Джордана

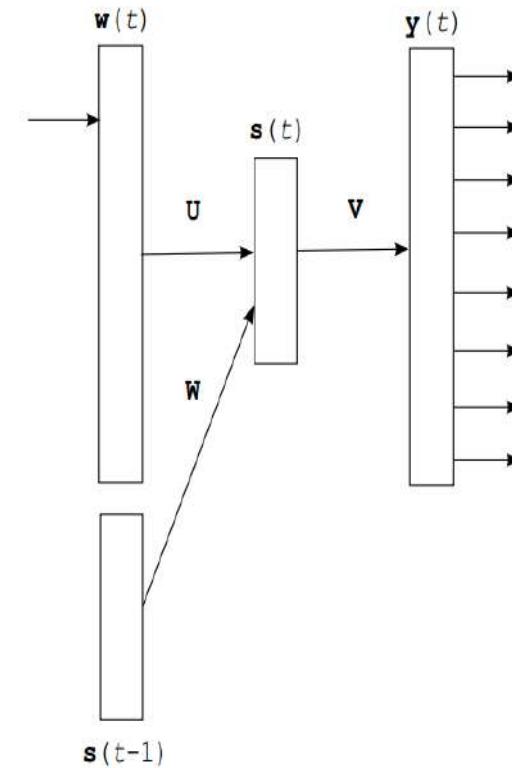
$$h_t = \sigma_h(W_h x_t + U_h y_{t-1} + b_h)$$

$$y_t = \sigma_y(W_y h_t + b_y)$$



Архитектура: простая RNN

- Входной слой, скрытый слой с рекуррентными соединениями и выходной слой
- В теории скрытый слой может обладать неограниченной памятью
- Также называется сетью Элмана (*Finding structure in time*, Elman 1990)



Архитектура: простая RNN

$$\mathbf{s}(t) = f(\mathbf{U}\mathbf{w}(t) + \mathbf{W}\mathbf{s}(t-1))$$

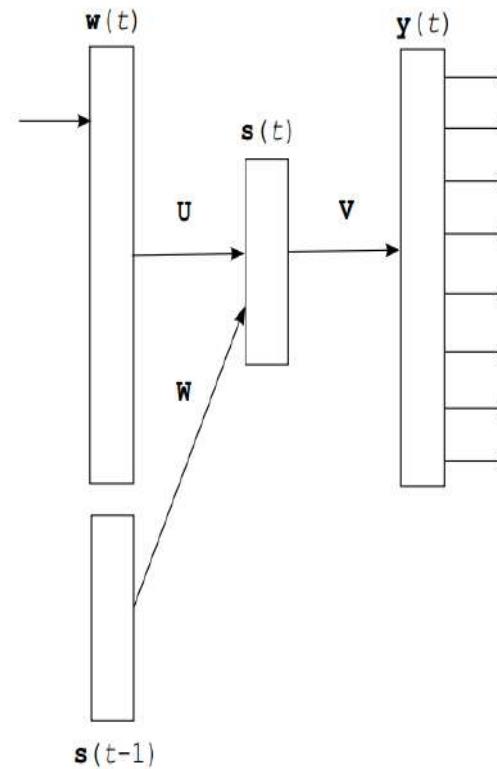
$$\mathbf{y}(t) = g(\mathbf{V}\mathbf{s}(t))$$

$f()$ чаще всего сигмоидальная
активационная функция:

$$f(z) = \frac{1}{1 + e^{-z}}$$

$g()$ чаще всего softmax:

$$g(z_m) = \frac{e^{z_m}}{\sum_k e^{z_k}}$$



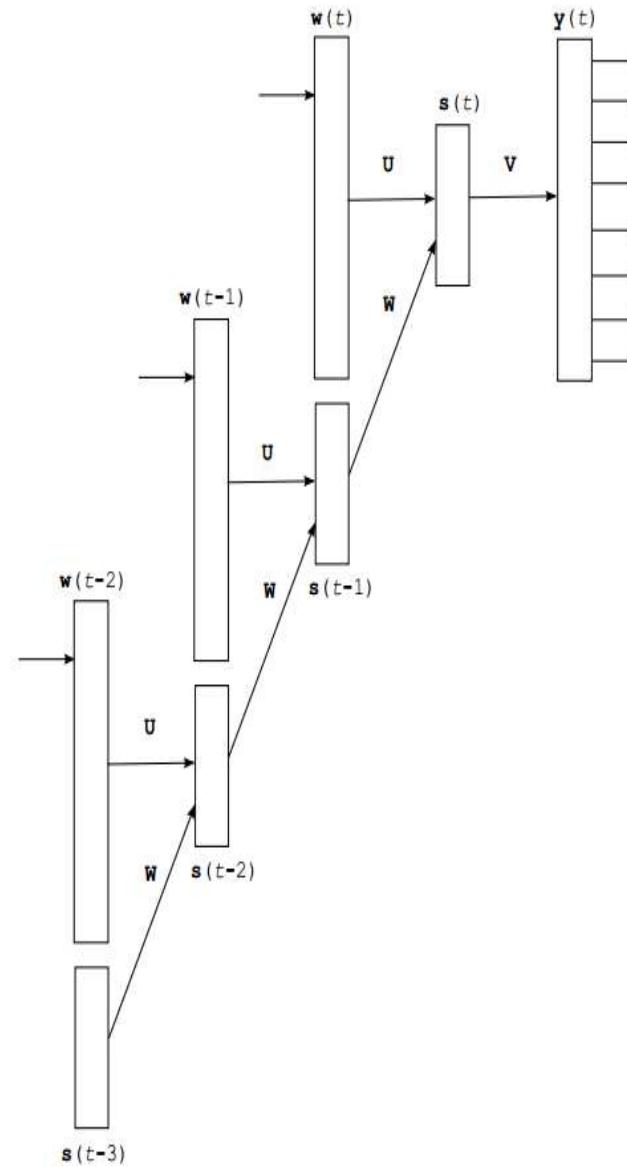
Обучение: обратное распространение по времени

(Backpropagation through time, BPTT)

- Как обучать рекуррентные сети?
- Выходное значение зависит от состояния скрытого слоя, который зависит от всех предыдущих состояний скрытого слоя (и, следовательно, всех предыдущих входов)
- Рекуррентная сеть может рассматриваться как (очень глубокая) сеть прямого распространения с общими весами

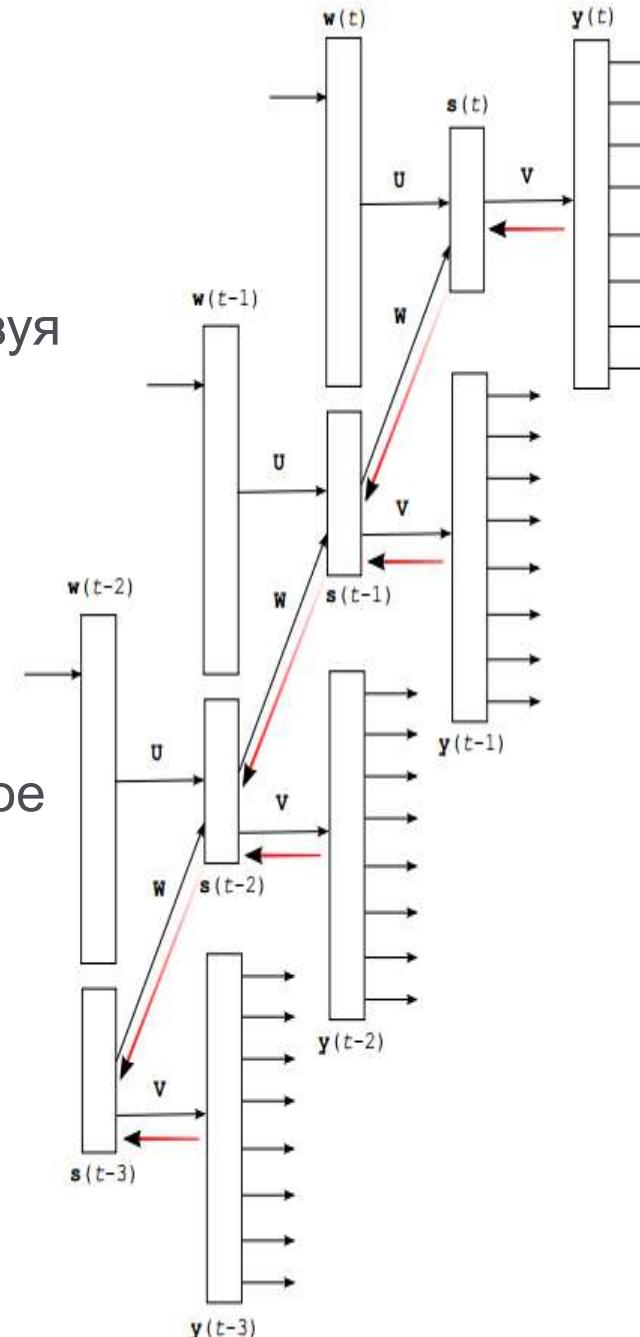
Обратное распространение по времени

- Идея заключается в том, что РНС разворачивается во времени
- Мы получаем глубокую нейронную сеть с общими весами \mathbf{U} и \mathbf{W}
- Часто бывает достаточно развернуть на несколько шагов (называется укороченным ВРТТ)



Обратное распространение по времени

- Мы тренируем развернутый РНС, используя обычное обр. распр-е + СГС
- На практике ограничивают количество шагов разворачивания до 5 – 10
- Эффективнее вычислить градиент после нескольких обучающих примеров (пакетное обучение)



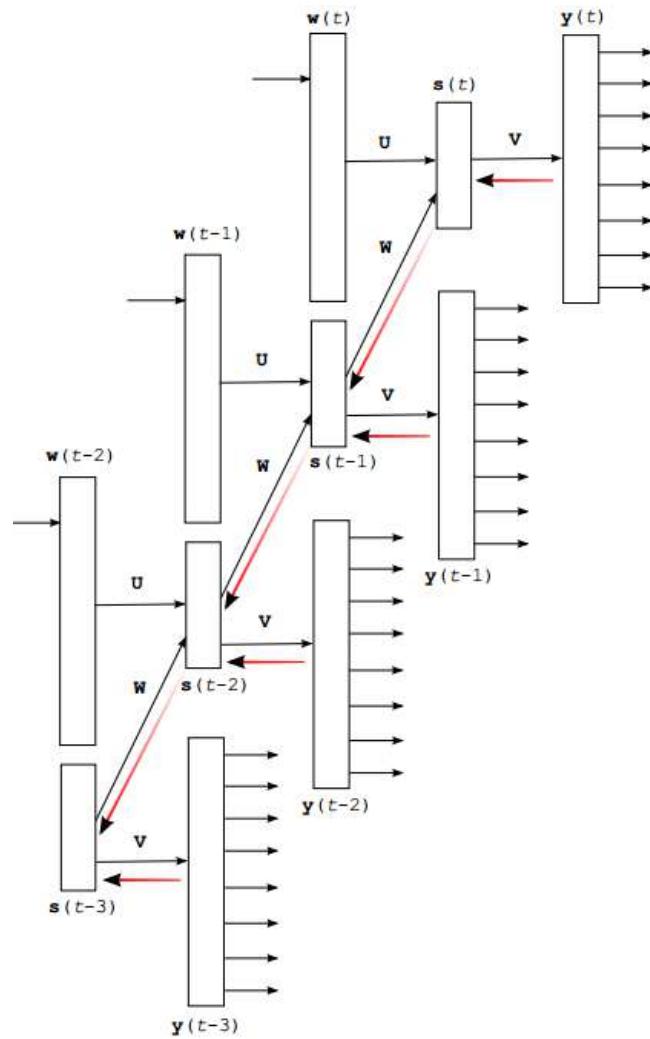
Исчезающие градиенты

- Когда мы распространяем градиенты назад во времени, обычно их величина быстро уменьшается: это называется «проблемой исчезающего градиента»
- На практике это означает, что изучение долгосрочных зависимостей в данных затруднительно для простой RNN архитектуры
- Специальные архитектуры RNN решают эту проблему:
 - Экспоненциальная память трассировки (Jordan 1987, Mozer 1989)
 - Долгая краткосрочная память (Hochreiter & Schmidhuber, 1997))
 - будут обсуждены во второй части этой лекции
- Множество научных теорий
 - лучше инициализировать рекуррентную матрицу и использовать импульс во время обучения
- Sutskever et.al.,: On The Importance of Initialization and Momentum in Deep Learning
 - изменять архитектуру

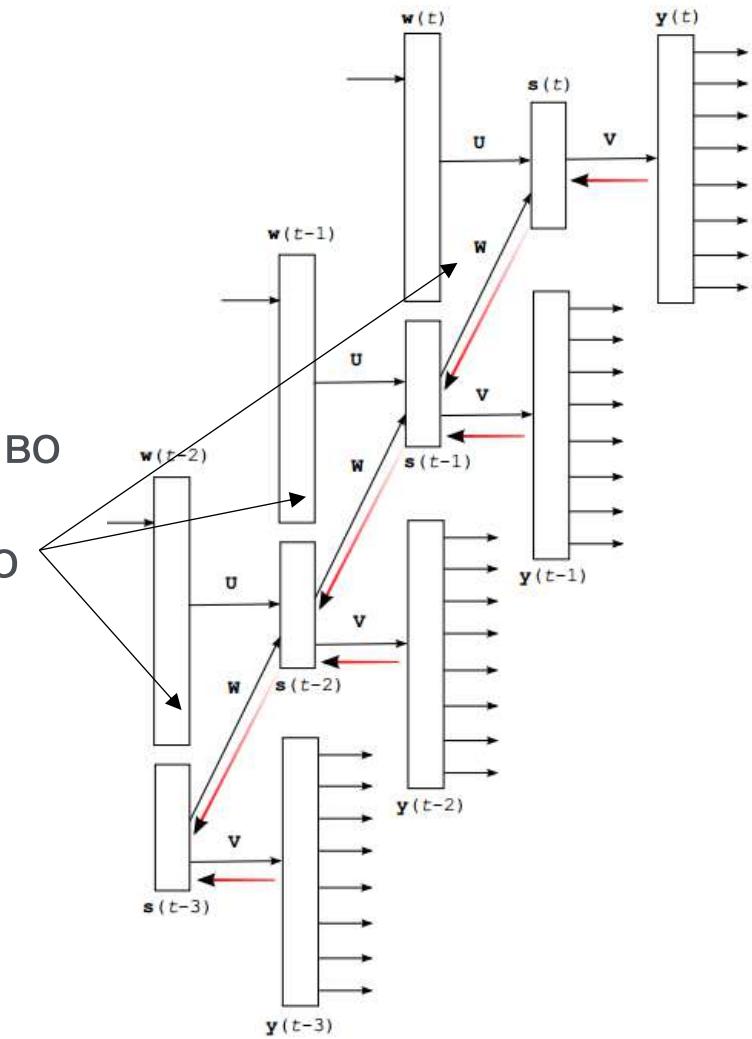
Взрывающиеся градиенты

- Иногда градиенты начинают экспоненциально возрастать во время обратного распространения через рекуррентные веса
- Бывает редко, но эффект может быть катастрофическим: огромные градиенты приведут к большому изменению весов и, таким образом, разрушат то, что было изучено до сих пор
- Одна из основных причин, по которым RNN должны были быть нестабильными
- Простое решение (впервые опубликован в RNNLM toolkit в 2010): обрезать или нормализовать значения градиентов, чтобы избежать огромных изменений весов

Обучение: Взрывающиеся градиенты



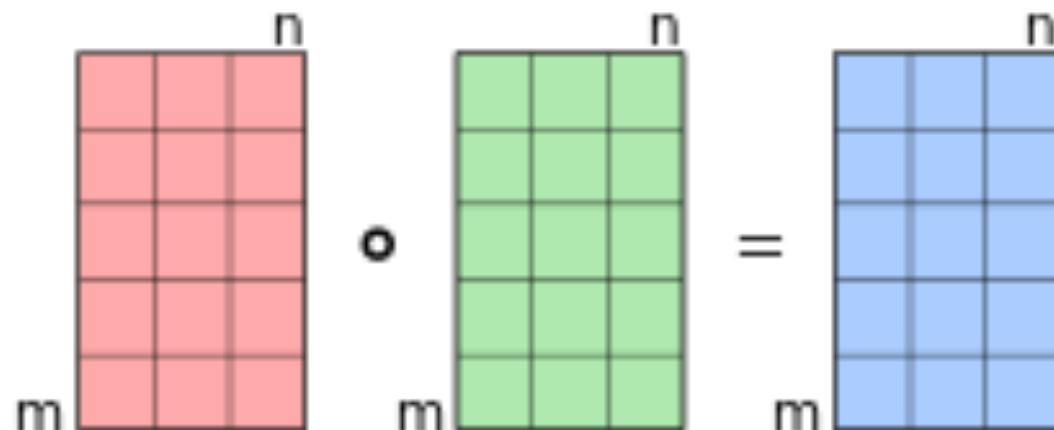
– Обрезка градиента во
время обратного
распространения по
времени



Произведение Адамара

- Произведение Адамара оперирует двумя матрицами одинаковой размерности и создаёт новую матрицу идентичной размерности.

$$(A \circ B)_{i,j} = (A)_{i,j} \cdot (B)_{i,j}$$

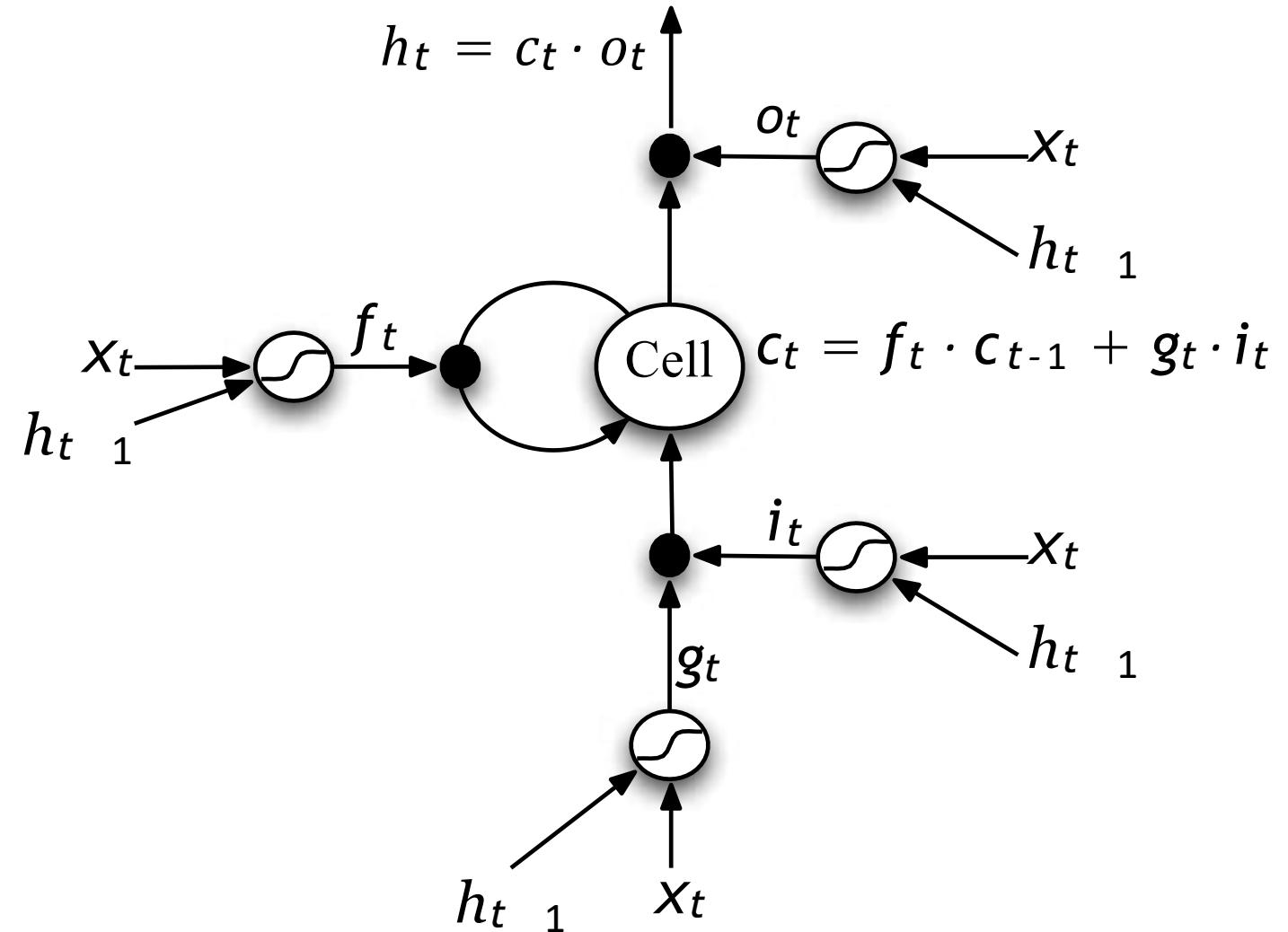


$$A \circ B = C$$

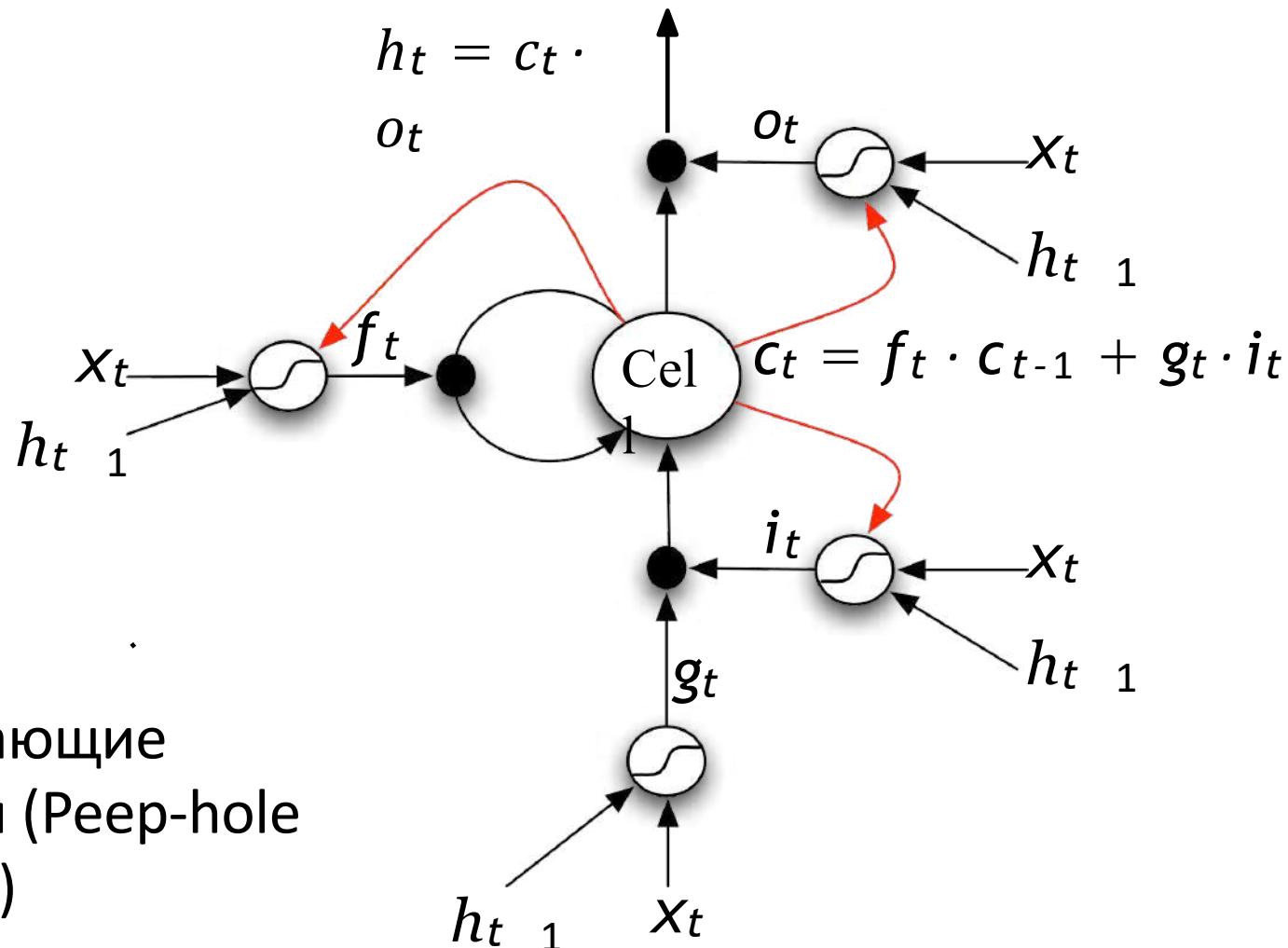
$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \circ \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} & a_{12}b_{12} & a_{13}b_{13} \\ a_{21}b_{21} & a_{22}b_{22} & a_{23}b_{23} \\ a_{31}b_{31} & a_{32}b_{32} & a_{33}b_{33} \end{bmatrix}$$

Долгая краткосрочная память (LSTM)

- Приобрела большую популярность для обработки временных рядов и текстов
- Использовалась в машинном переводе до появления трансформеров
- Имеются явные «ячейки» памяти для хранения кратковременных активаций, наличие дополнительных вентиляй частично устраняет проблему исчезающего градиента
- Многослойные версии показали, что они хорошо работают над задачами, которые имеют среднесрочные зависимости

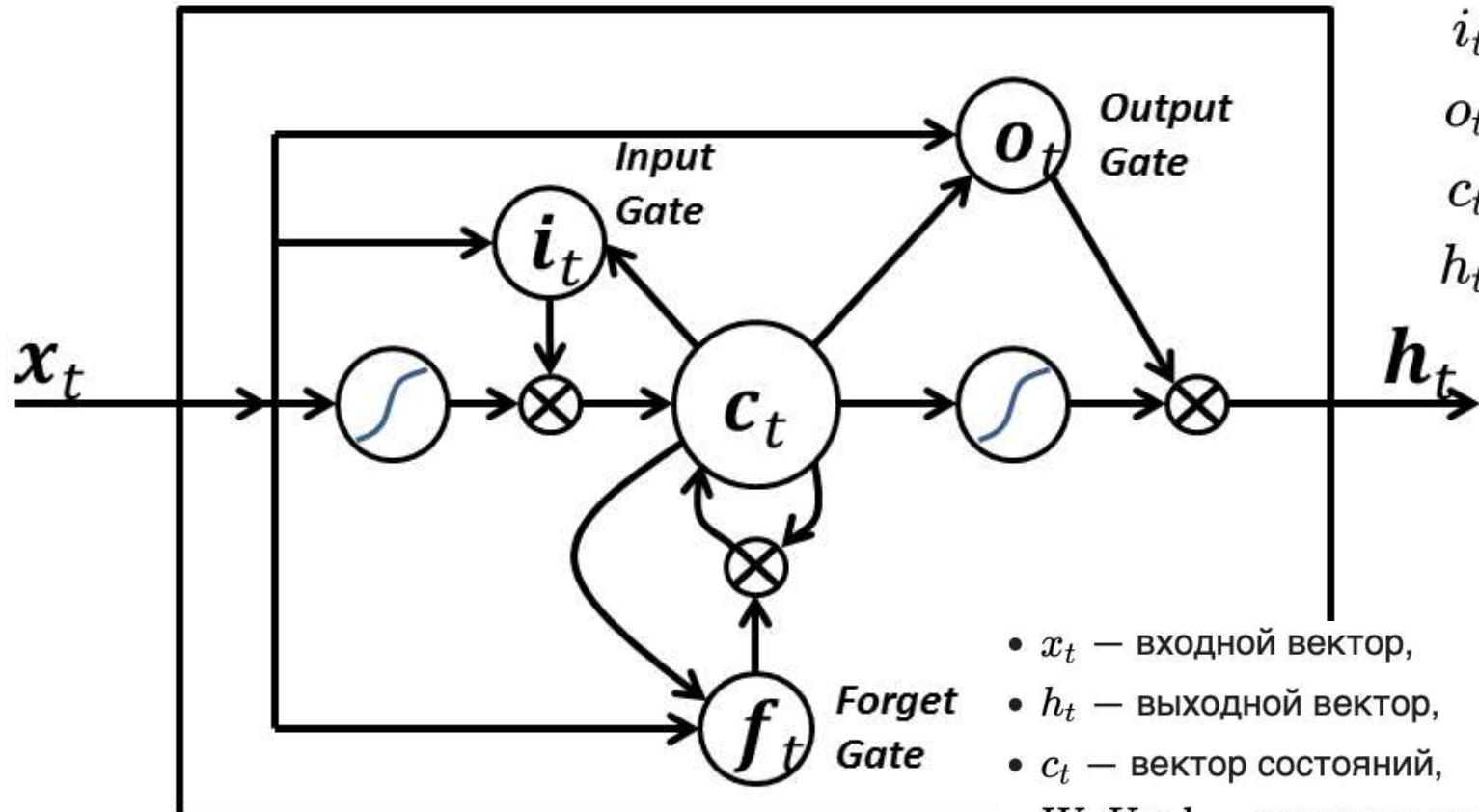


Долгая краткосрочная память (LSTM) с «глазками»



Подглядывающие
соединения (Peep-hole
connections)

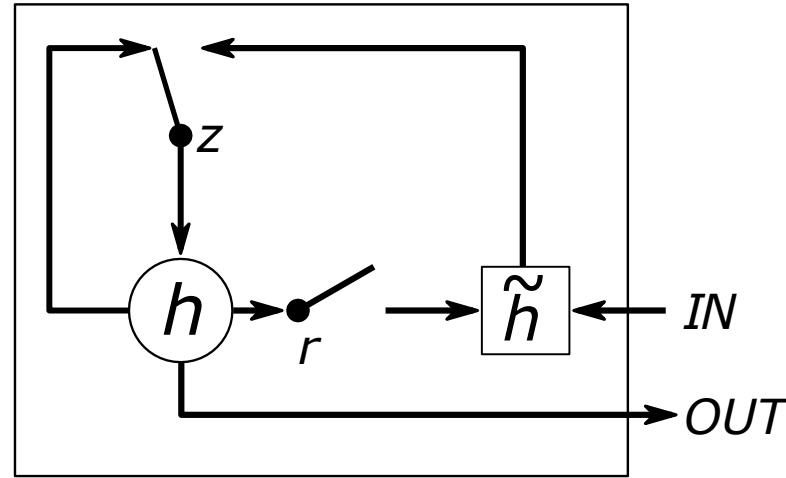
LSTM



$$f_t = \sigma_g(W_f x_t + U_f c_{t-1} + b_f)$$
$$i_t = \sigma_g(W_i x_t + U_i c_{t-1} + b_i)$$
$$o_t = \sigma_g(W_o x_t + U_o c_{t-1} + b_o)$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + b_c)$$
$$h_t = o_t \circ \sigma_h(c_t)$$

- x_t — входной вектор,
- h_t — выходной вектор,
- c_t — вектор состояний,
- W, U и b — матрицы параметров и вектор,
- f_t, i_t и o_t — векторы вентиляй,
 - f_t — вектор вентиля забывания, вес запоминания старой информации,
 - i_t — вектор входного вентиля, вес получения новой информации,
 - o_t — вектор выходного вентиля, кандидат на выход.

Рекуррентные нейрон с вентилями (Gated recurrent unit, GRU)



- Вентиль обновления: $z_t^j = \sigma(W_z \mathbf{x}_t + U_z \mathbf{h}_{t-1})^j$.
- Вентиль перезаписи: $r_t^j = \sigma(W_r \mathbf{x}_t + U_r \mathbf{h}_{t-1})^j$.
- Кандидат на активацию: $\tilde{h}_t^j = \tanh(W \mathbf{x}_t + U(\mathbf{r}_t * \mathbf{h}_{t-1}))^j$,

$$h_t^j = (1 - z_t^j) h_{t-1}^j + z_t^j \tilde{h}_t^j,$$

GRU

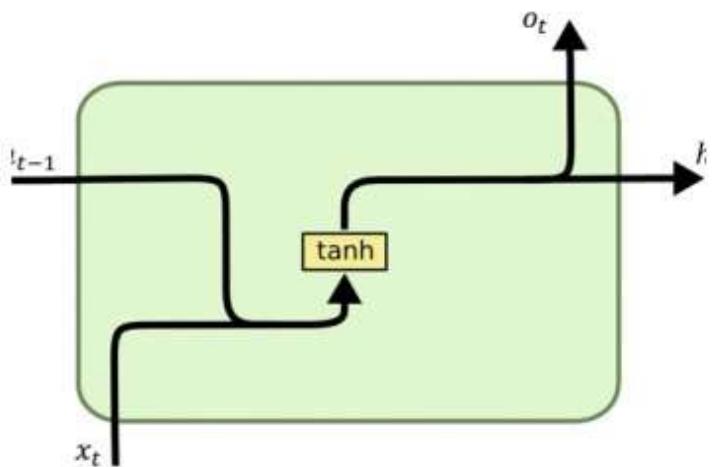
$$z_t = \sigma_g(W_z x_t + U_z h_{t-1} + b_z)$$

$$r_t = \sigma_g(W_r x_t + U_r h_{t-1} + b_r)$$

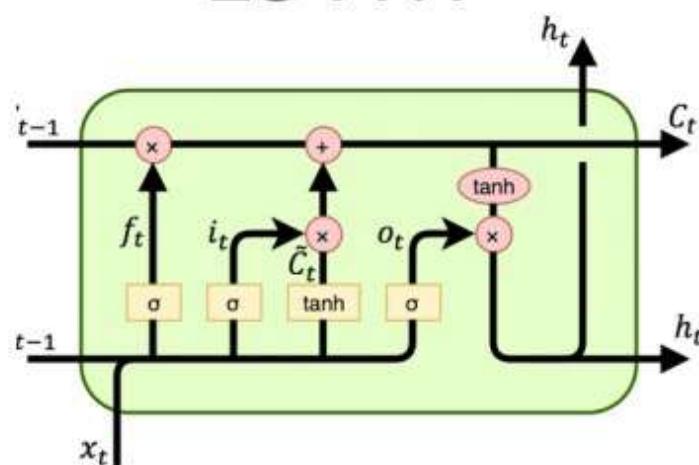
$$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \sigma_h(W_h x_t + U_h (r_t \circ h_{t-1}) + b_h)$$

- x_t : входной вектор
- h_t : выходной вектор
- z_t : вектор вентиля обновления
- r_t : вектор вентиля сброса
- W , U и b : матрицы параметров и вектор

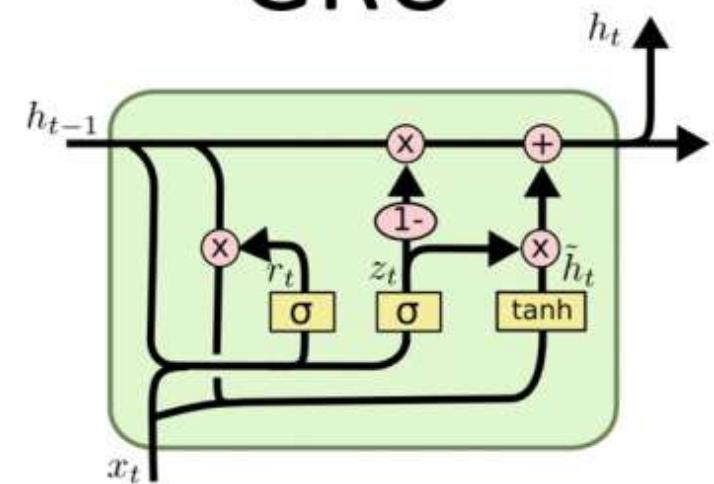
RNN



LSTM

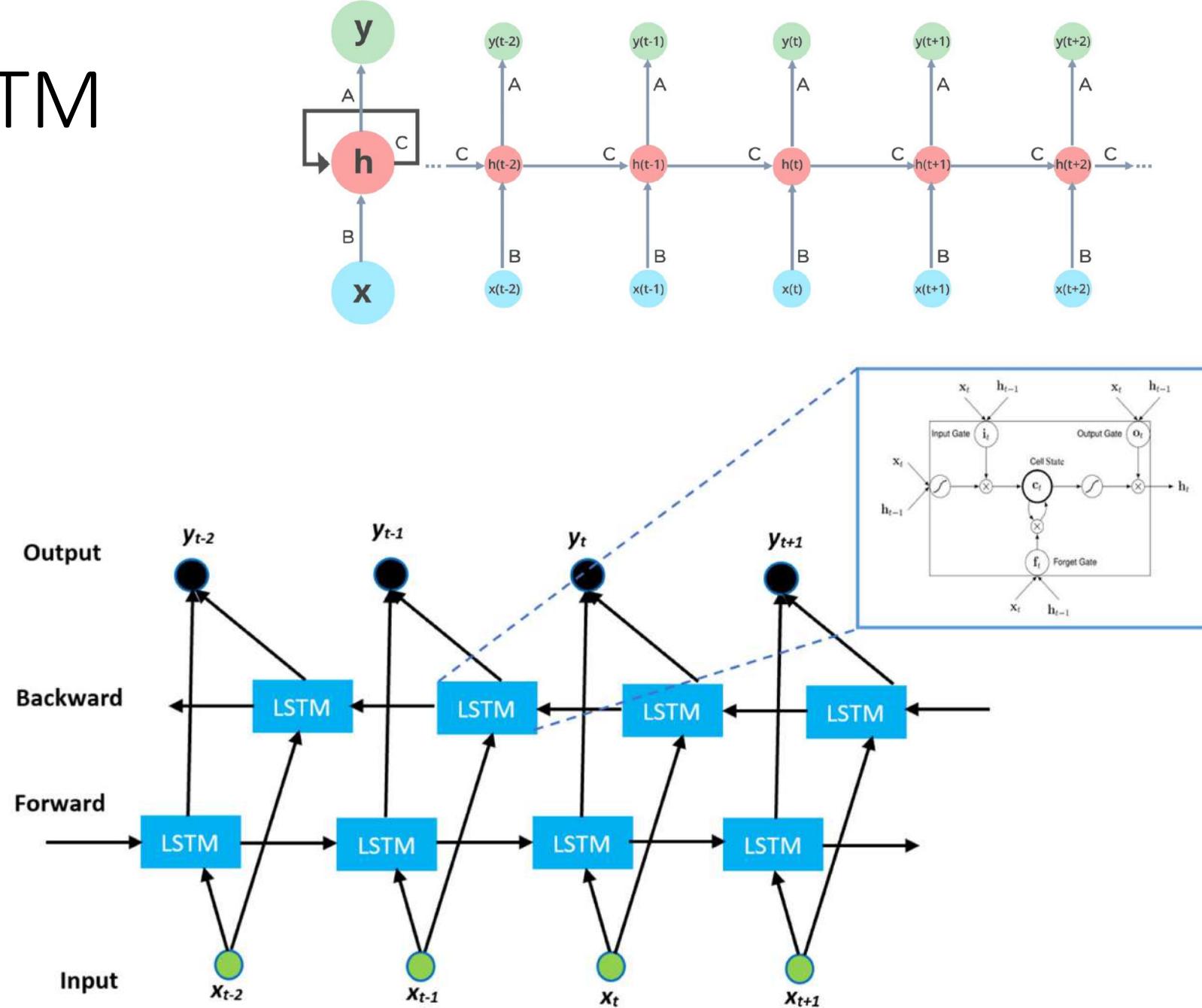


GRU



Bidirectional LSTM

- В простой RNN или LSTM мы каждый раз даем на вход следующее значение
- В двунаправленной сети у нас уже две ячейки: одна предсказывает значение, зная все предыдущие, а вторая «читает» с конца
- Двунаправленная сеть используется в известной языковой модели ELMo



LSTM из практики

```
def __init__(self, num_features, input_size, hidden_size, num_layers,
            bidirectional=True, p=0.4):
    super(LSTM, self).__init__()

    self.num_features = num_features
    self.num_layers = num_layers
    self.input_size = input_size
    self.hidden_size = hidden_size
    self.bidirectional = bidirectional

    self.lstm = nn.LSTM(input_size=input_size,
                        hidden_size=hidden_size,
                        bidirectional=bidirectional,
                        num_layers=num_layers,
                        batch_first=True)
    self.dropout = nn.Dropout(p)

    self.fc = nn.Linear(2*hidden_size if bidirectional else hidden_size, num_features)
```

```
num_features = 1
input_size = 1
hidden_size = 16
num_layers = 2
bidirectional = True
dropout_rate = 0.2
```

Наша рекуррентная сеть **двунаправленная**

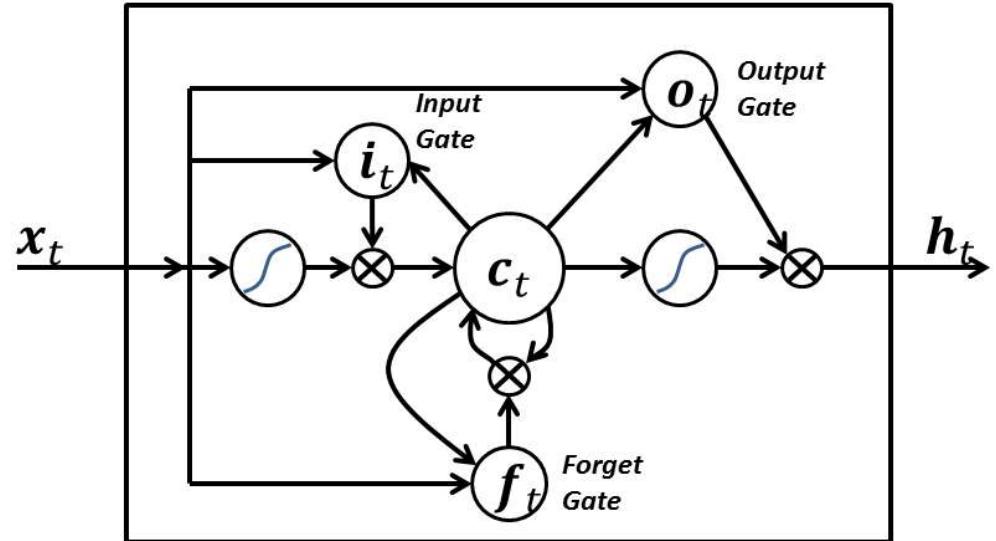
Состоит из:

- двух слоев LSTM
- слой dropout
- полносвязный выходной слой

LSTM из практики

```
self.lstm = nn.LSTM(input_size=input_size,
                     hidden_size=hidden_size,
                     bidirectional=bidirectional,
                     num_layers=num_layers,
                     batch_first=True)
self.dropout = nn.Dropout(p)

self.fc = nn.Linear(2*hidden_size if bidirectional
lstm.weight_ih_10 torch.Size([64, 1])
lstm.weight_hh_10 torch.Size([64, 16])
lstm.bias_ih_10 torch.Size([64])
lstm.bias_hh_10 torch.Size([64])
lstm.weight_ih_10_reverse torch.Size([64, 1])
lstm.weight_hh_10_reverse torch.Size([64, 16])
lstm.bias_ih_10_reverse torch.Size([64])
lstm.bias_hh_10_reverse torch.Size([64])
lstm.weight_ih_11 torch.Size([64, 32])
lstm.weight_hh_11 torch.Size([64, 16])
lstm.bias_ih_11 torch.Size([64])
lstm.bias_hh_11 torch.Size([64])
lstm.weight_ih_11_reverse torch.Size([64, 32])
lstm.weight_hh_11_reverse torch.Size([64, 16])
lstm.bias_ih_11_reverse torch.Size([64])
lstm.bias_hh_11_reverse torch.Size([64])
fc.weight torch.Size([1, 32])
fc.bias torch.Size([1])
```



$$i_t = \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi})$$

$$f_t = \sigma(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf})$$

$$g_t = \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg})$$

$$o_t = \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho})$$

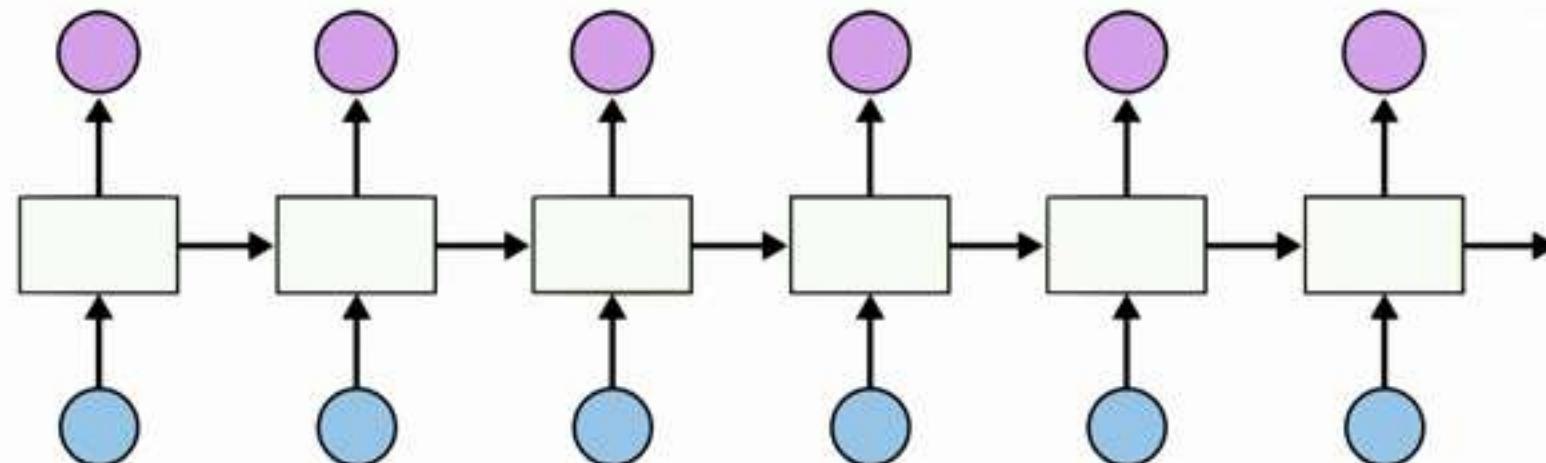
$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$

$$h_t = o_t \odot \tanh(c_t)$$

- В PyTorch для каждой ячейки LSTM обучаются 4 группы параметров: `weight_i`, `weight_h`, `bias_i`, `bias_h`
- Каждая группа содержит по 4 части для 3 вентиляй и состояния
- В задании 2 слоя ячеек и в каждом ячейки прямого и обратного распространения

RNN для генерации текста

- Рекуррентная нейронная сеть учитывает состояние ячейки для генерации нового символа
- Здесь простой пример для генерации символов
- Обычно генерируют слова или токены



Обработка текста

“A dog barked at a cat.”

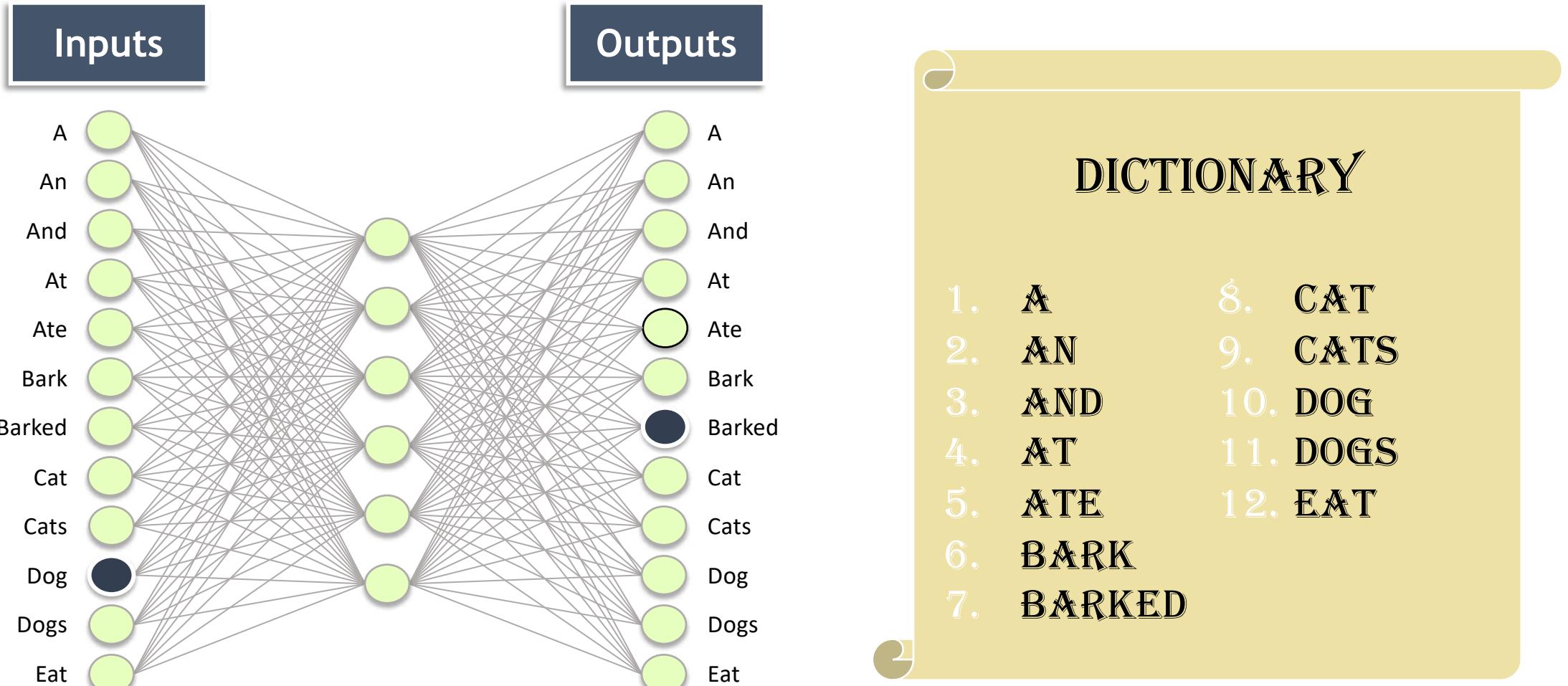
[1, 10, 7, 4, 1, 8]

- Составляем словарь слов – нумеруем все уникальные слова из нашего текста
- Ставим в соответствие каждому уникальному слову какой-то нейрон (входной или выходной)

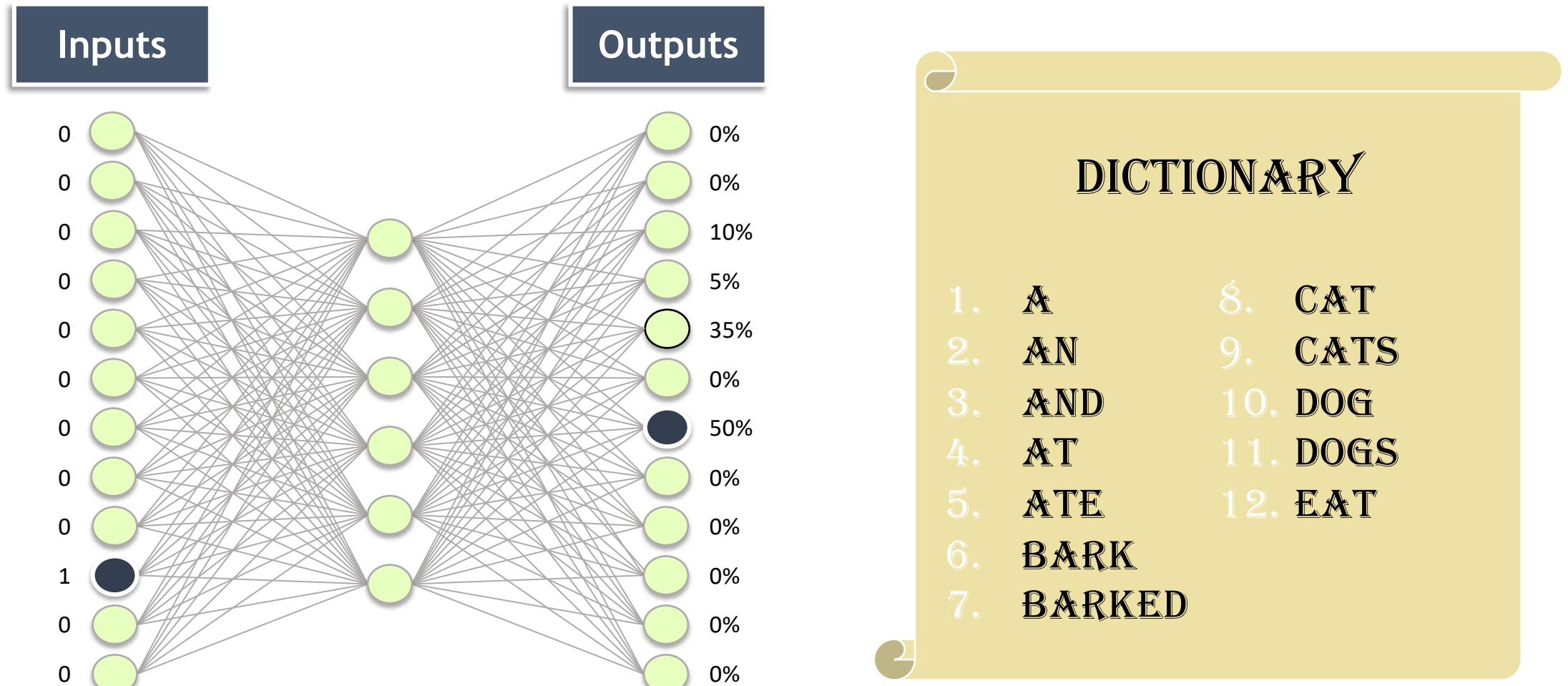
DICTIONARY

1.	A	8.	CAT
2.	AN	9.	CATS
3.	AND	10.	DOG
4.	AT	11.	DOGS
5.	ATE	12.	EAT
6.	BARK		
7.	BARKED		

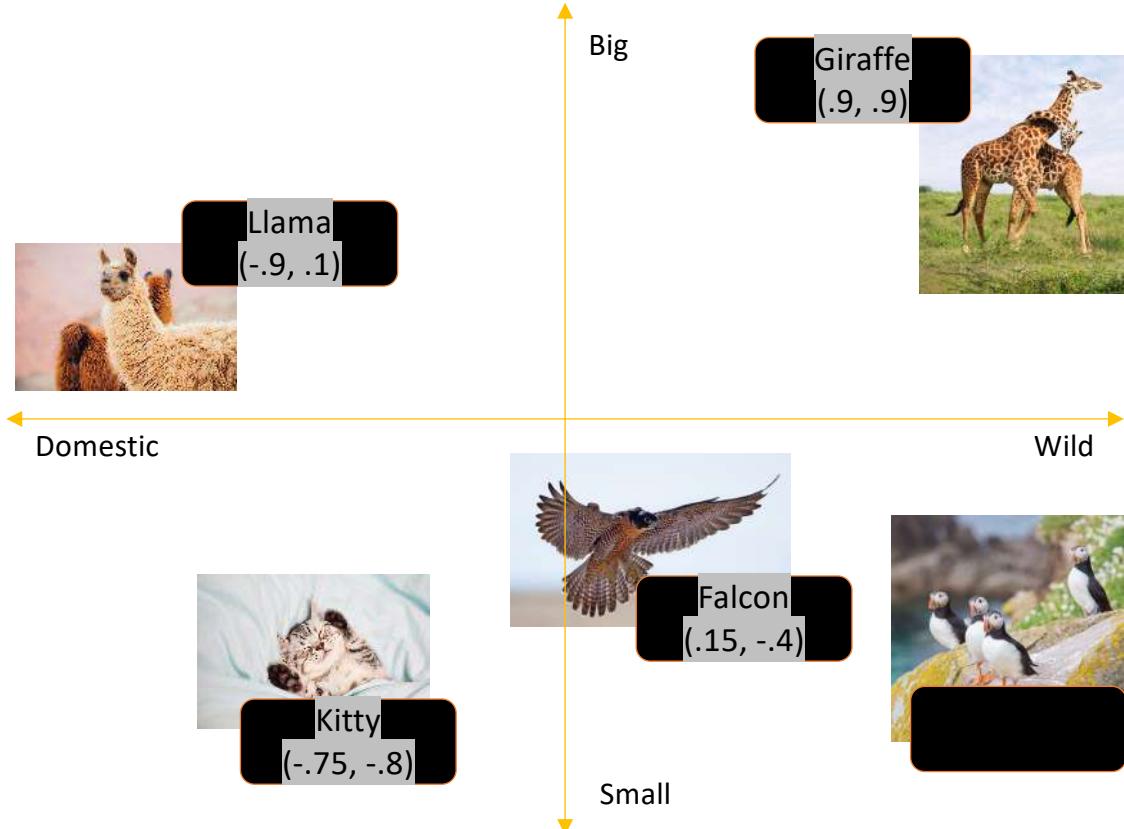
Полносвязная сеть для классификации



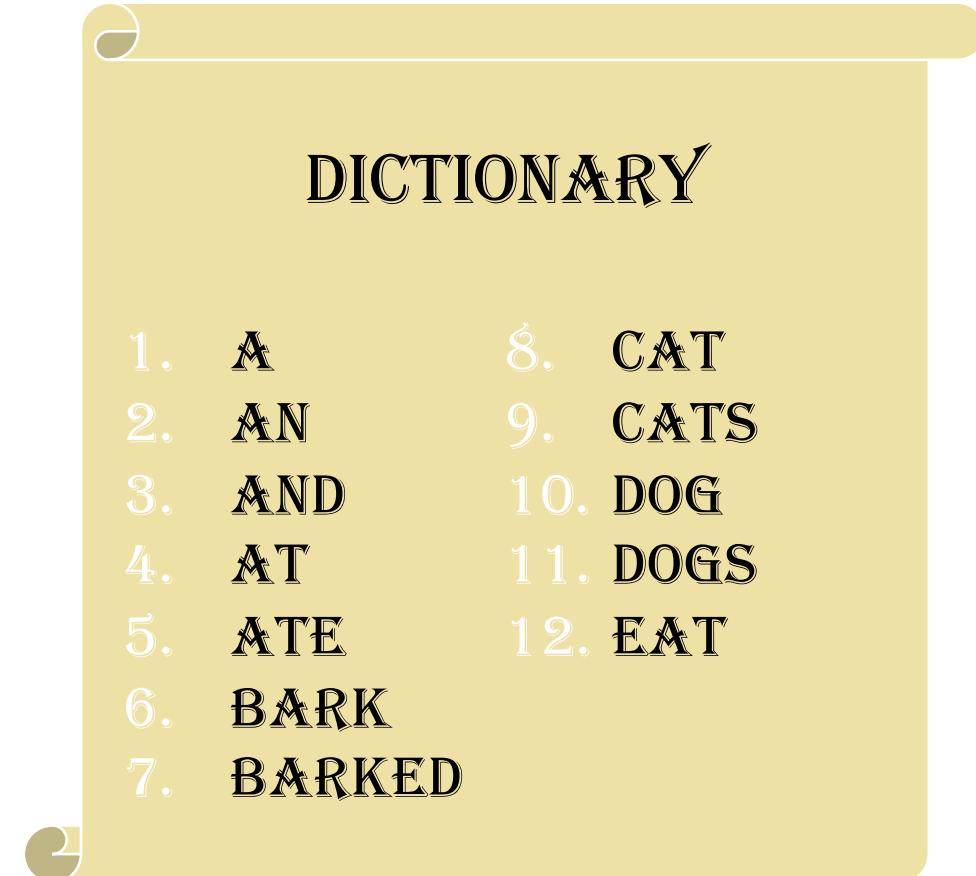
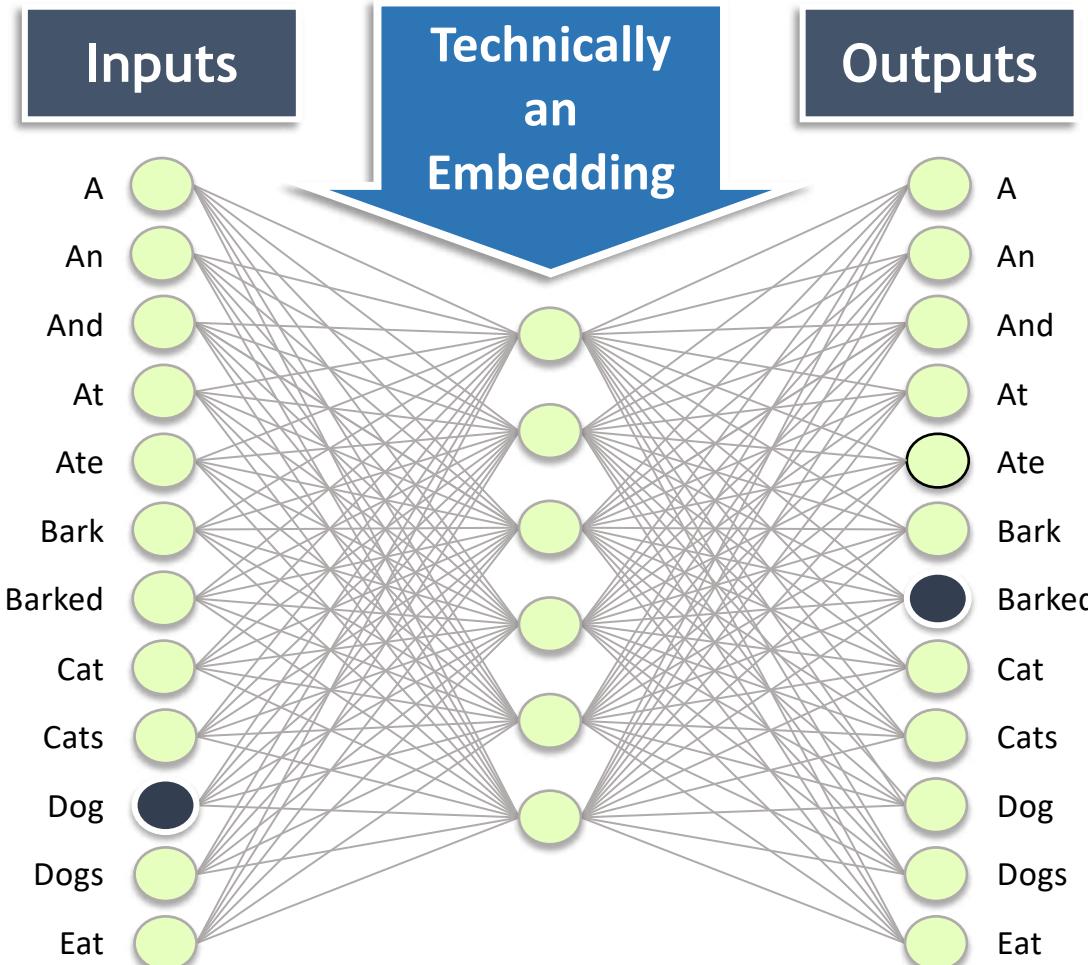
Полносвязная сеть для классификации



Замена словаря на embedding



Добавляем embedding в полносвязную сеть



Recurrent Neural Networks

- Усложним задачу: теперь в обоих случаях есть одно и то же слово "say" и сеть должна запомнить, что было до этого
- Для этого будем использовать RNN

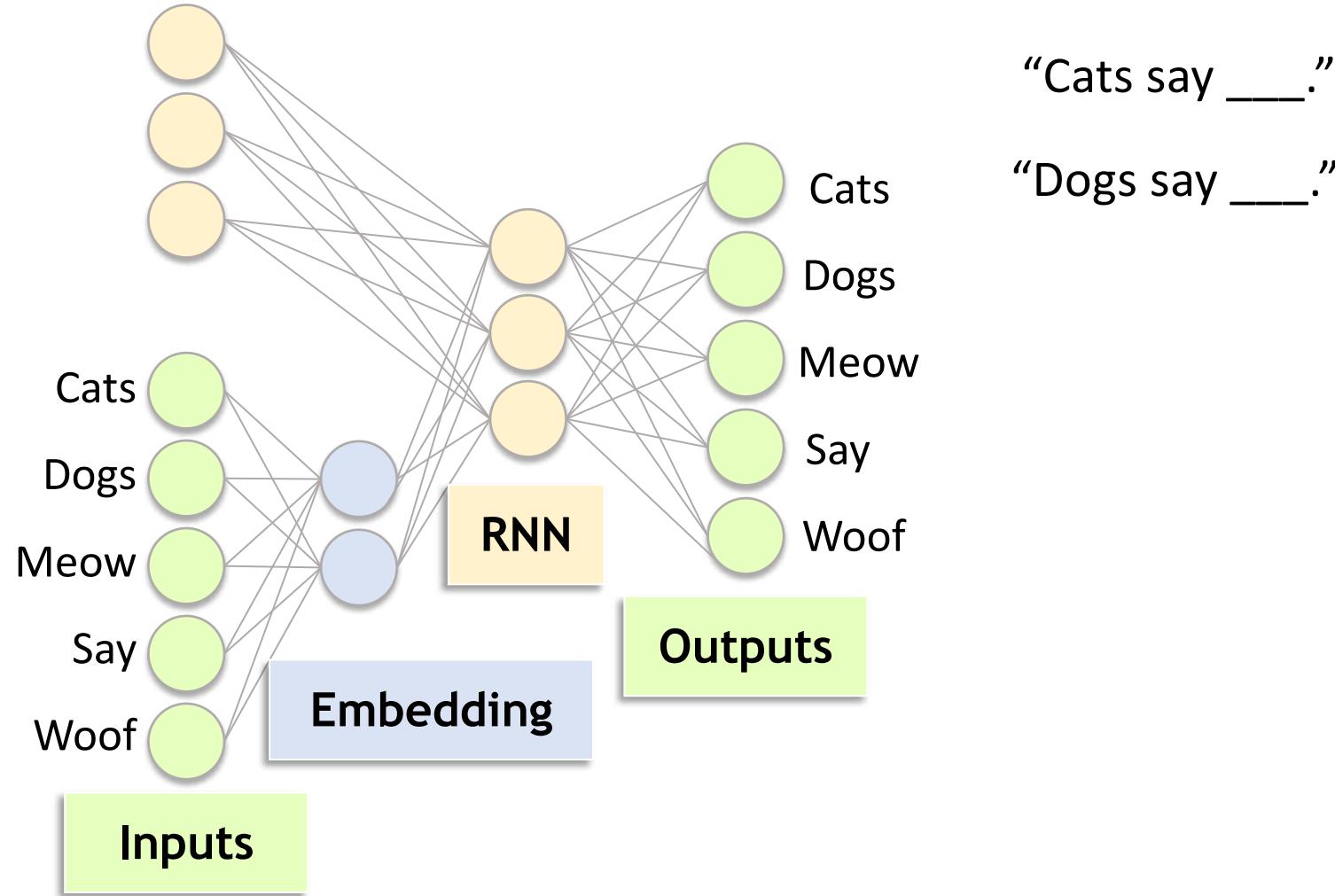
"Cats say ____."

"Dogs say ____."

DICTIONARY

1. CATS
2. DOGS
3. MEOW
4. SAY
5. WOOF

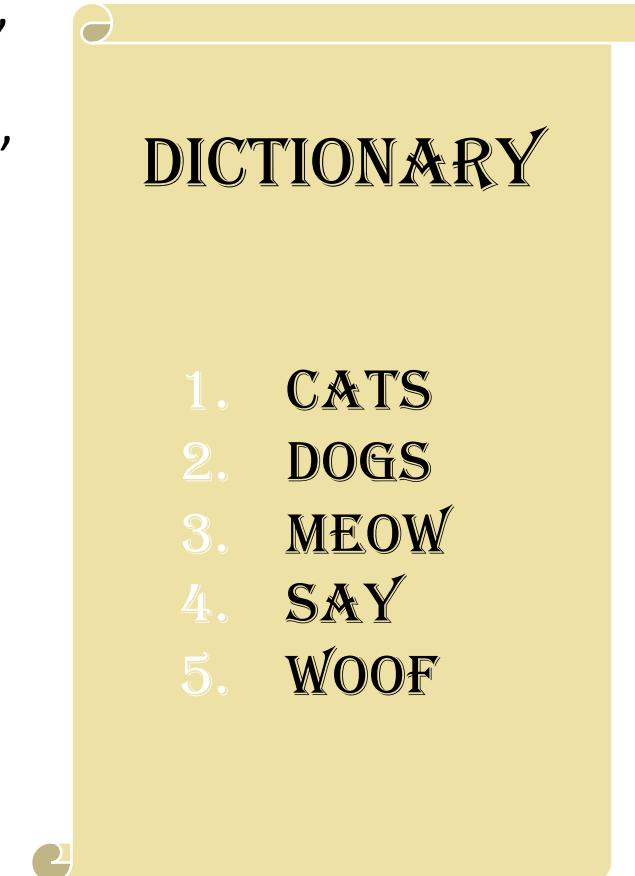
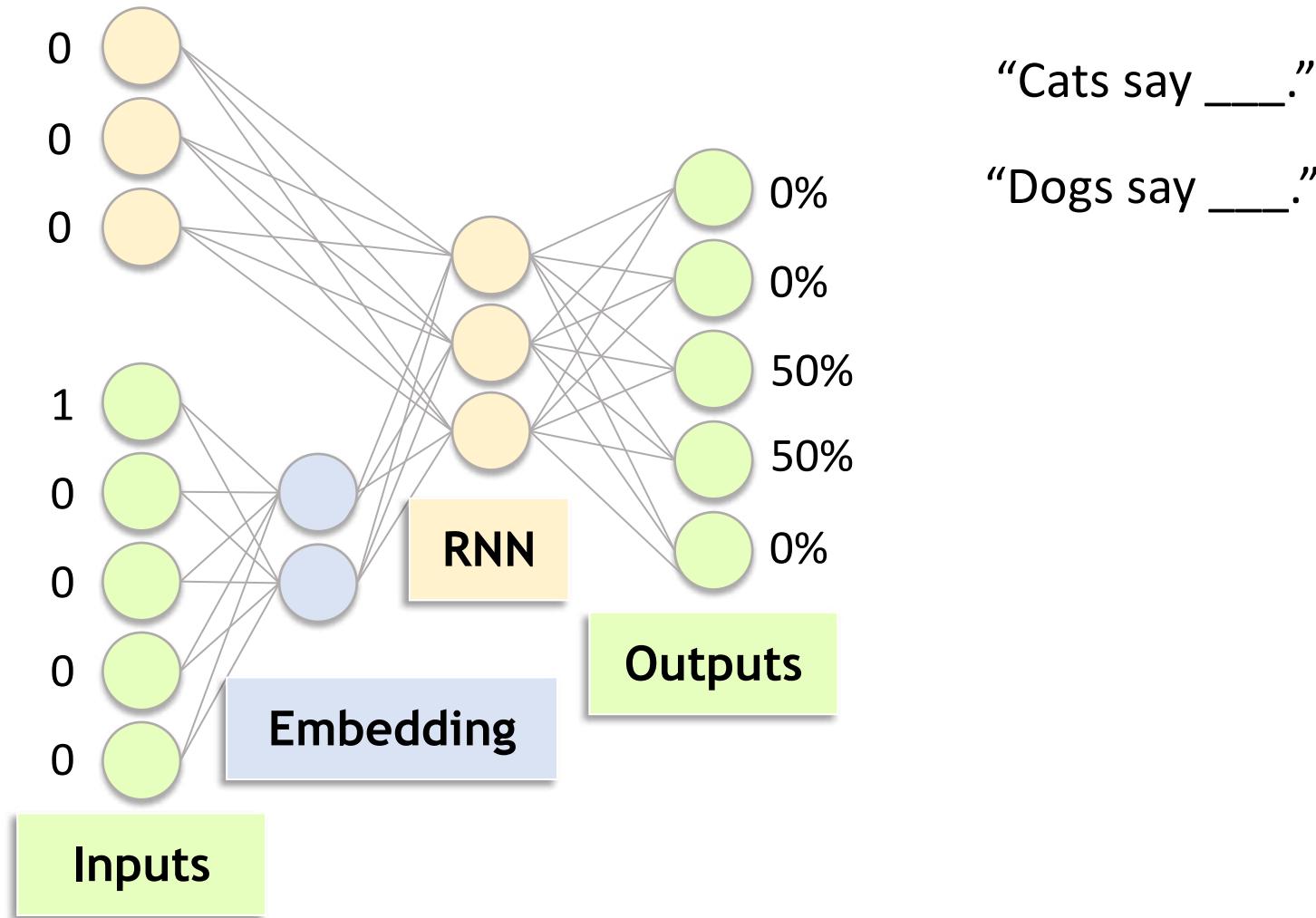
RNN для генерации текста



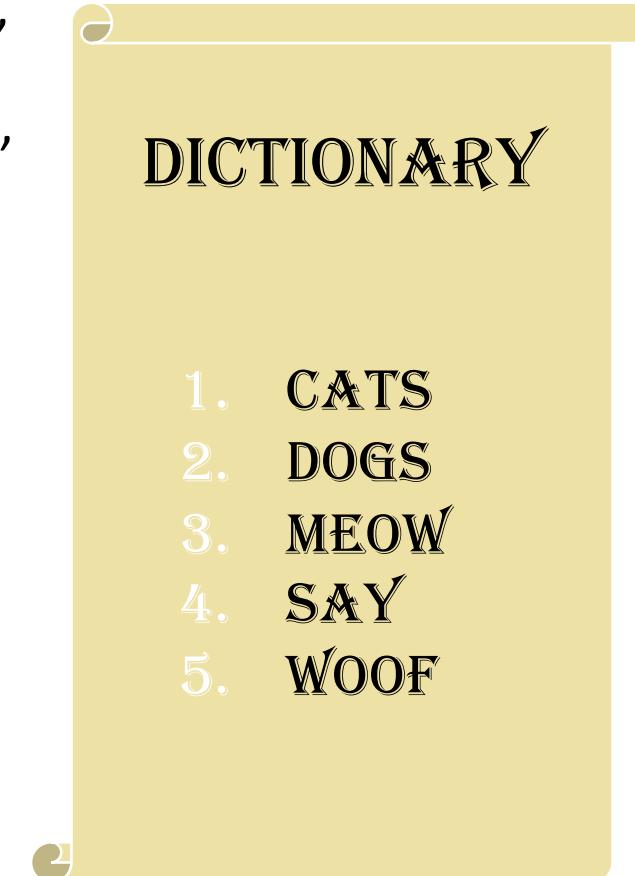
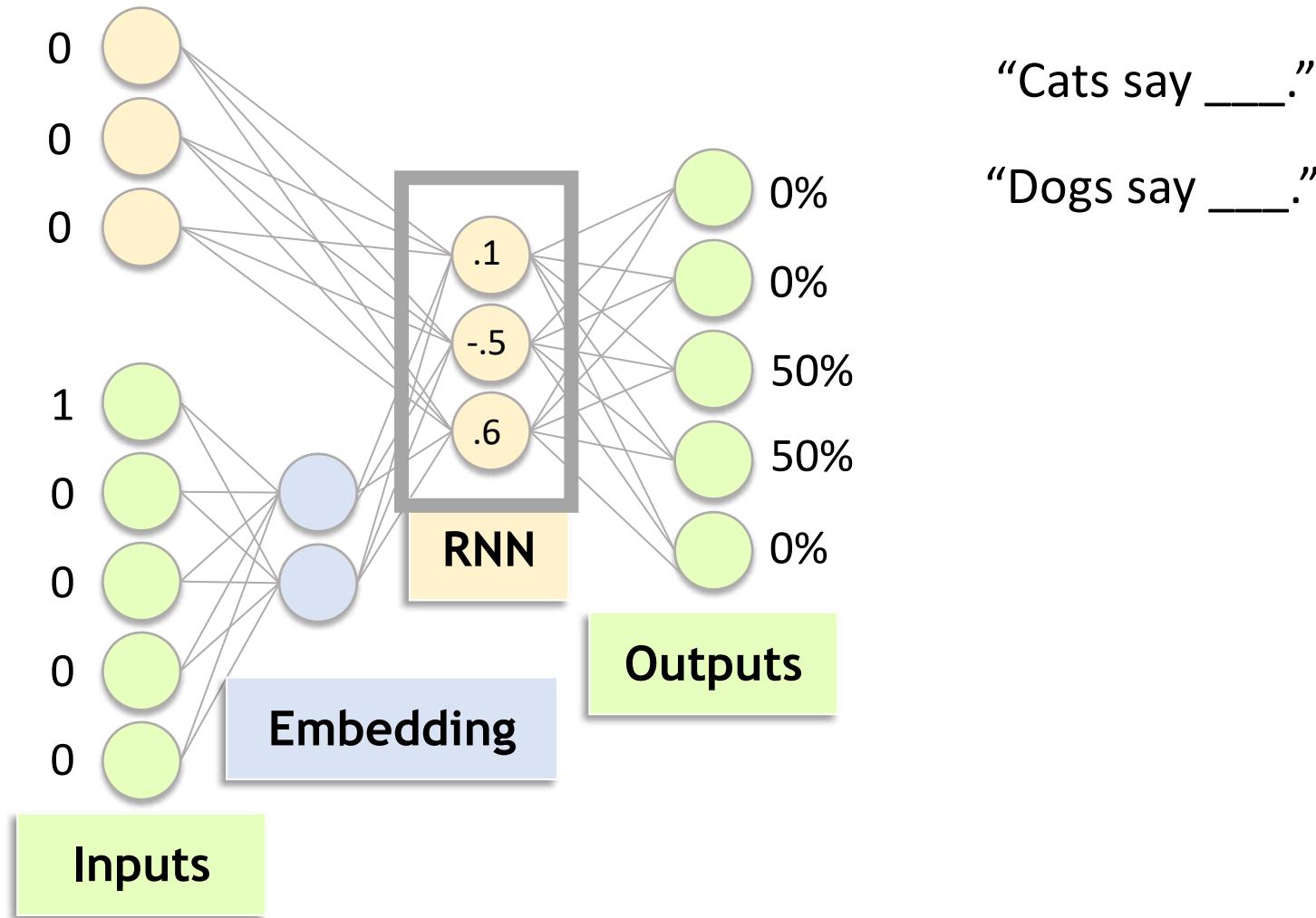
DICTIONARY

1. CATS
2. DOGS
3. MEOW
4. SAY
5. WOOF

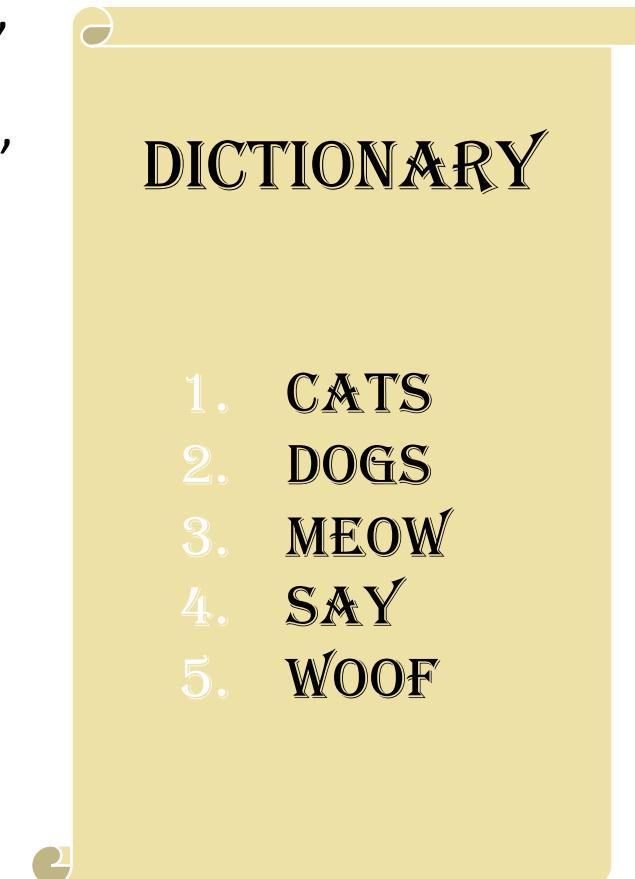
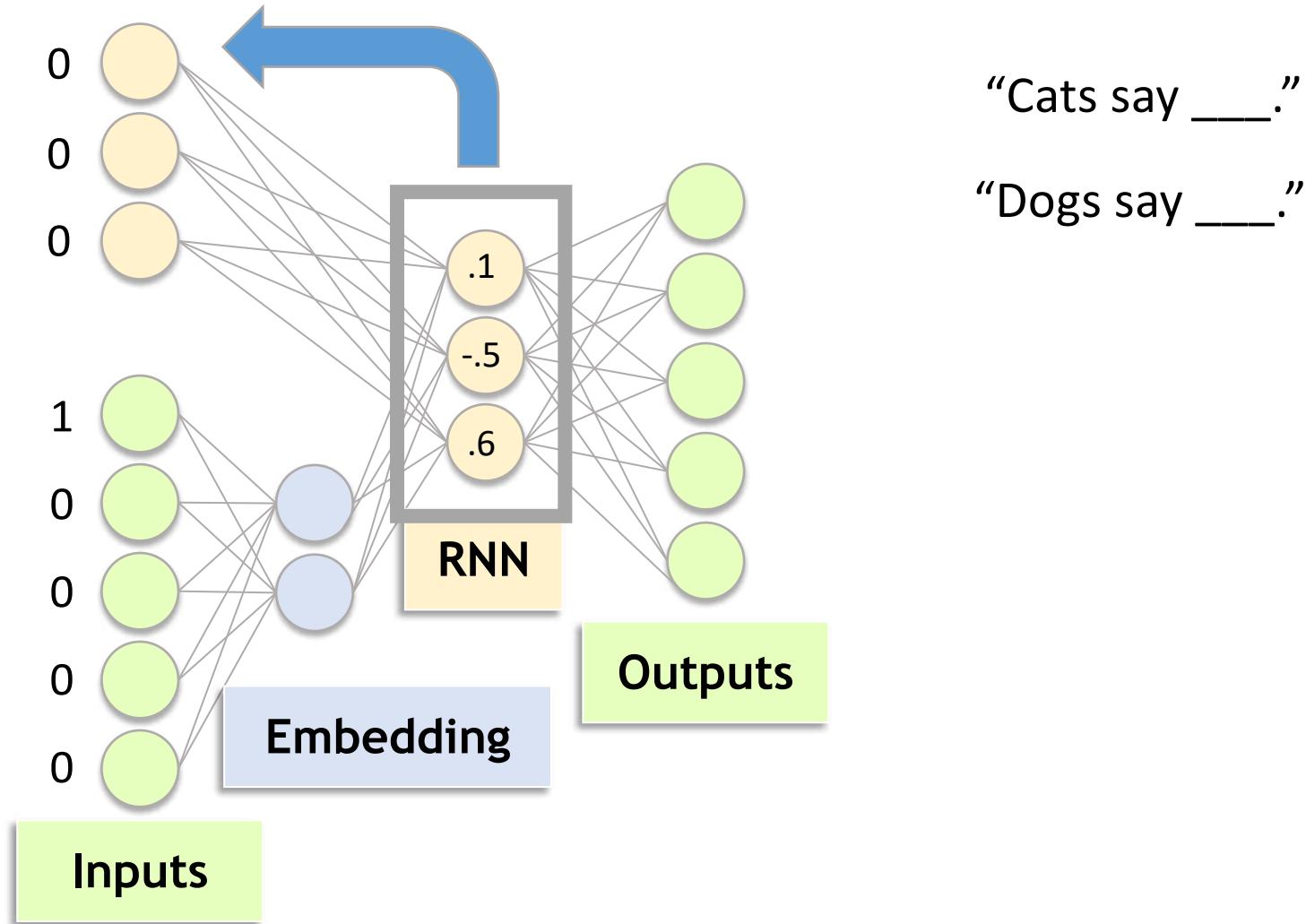
Начальное слово и состояние



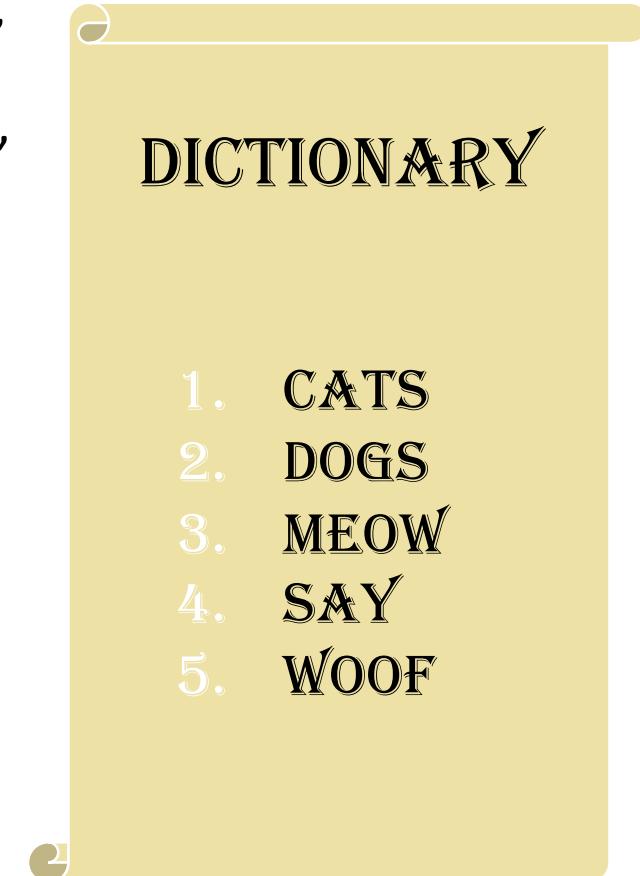
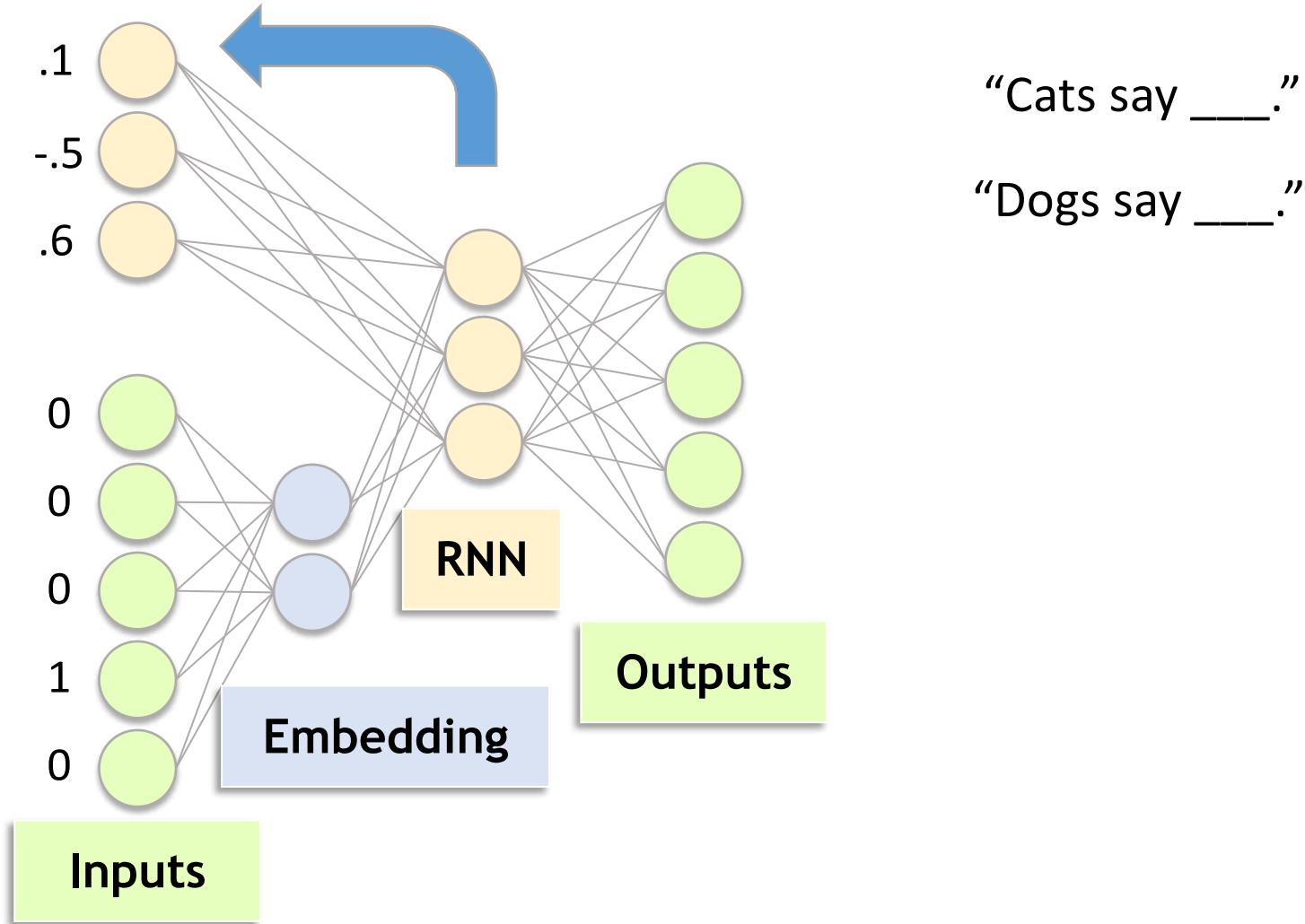
Вычисление нового состояния



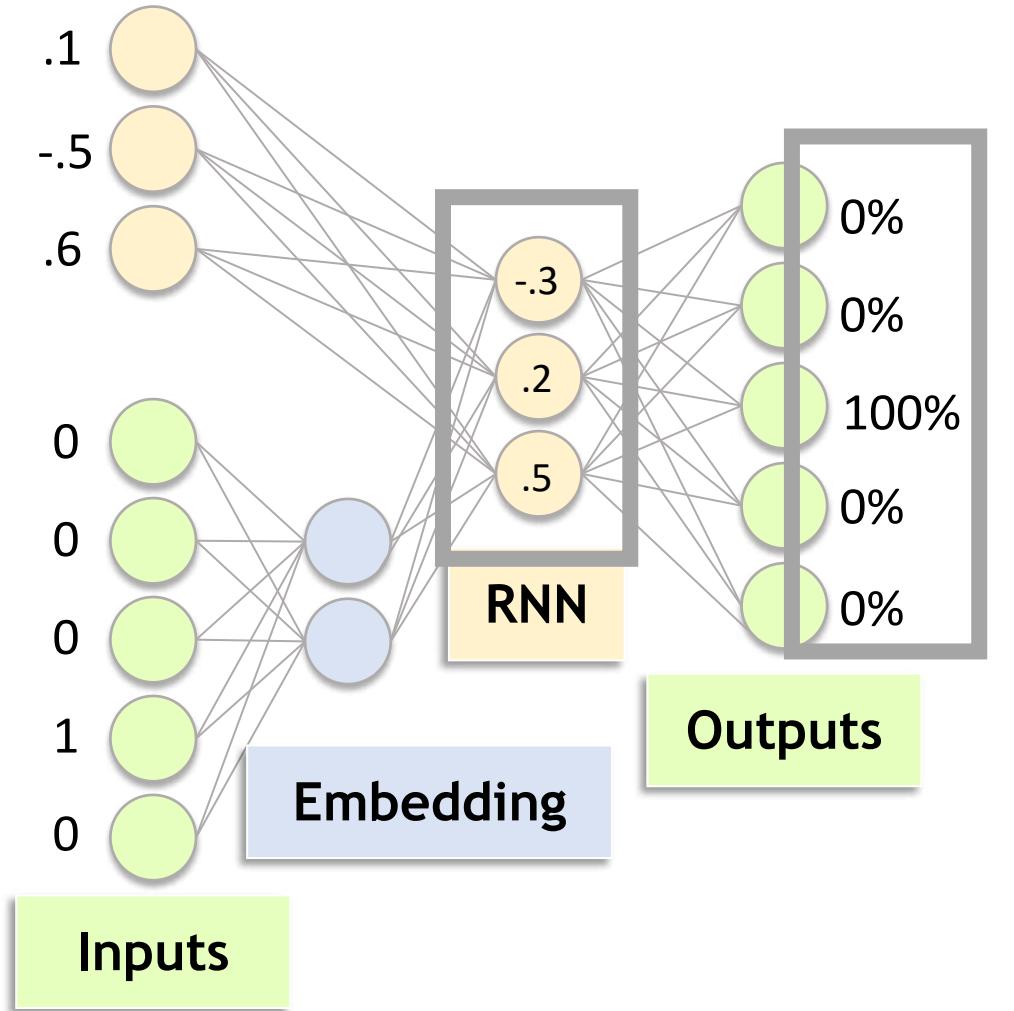
Запоминаем состояние



Новое слово и старое состояние



Получение ответа



"Cats say ____."

"Dogs say ____."

DICTIONARY

1. CATS
2. DOGS
3. MEOW
4. SAY
5. WOOF

Seq2seq

- Для машинного перевода
- Состоит из двух частей, например две LSTM
- Можно добавить внимание

