

Deep Learning

Нейронные сети и их приложения



Введение

О чём мы хотим рассказать:

- Что на самом деле скрывается за модным понятием Deep Learning
- Как устроены нейронные сети и то, как построен процесс их обучения
- Современные архитектуры нейронных сетей
- Современные фреймворки
- И многое другое

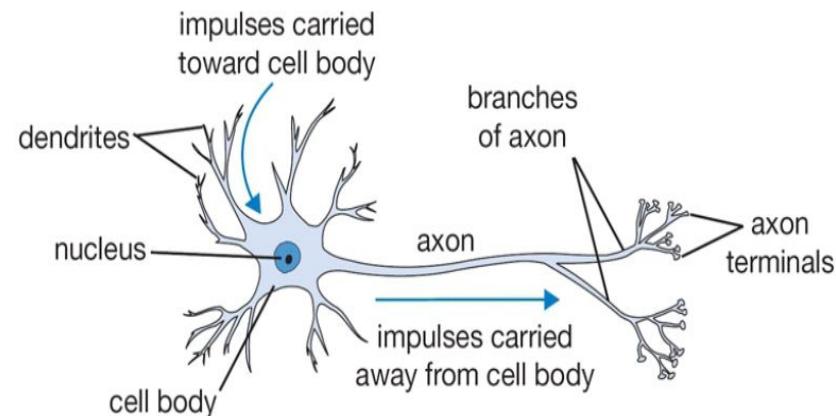
Предпосылки развития

Отличия DL от классических методов ML

Сфера применения

Предпосылки развития

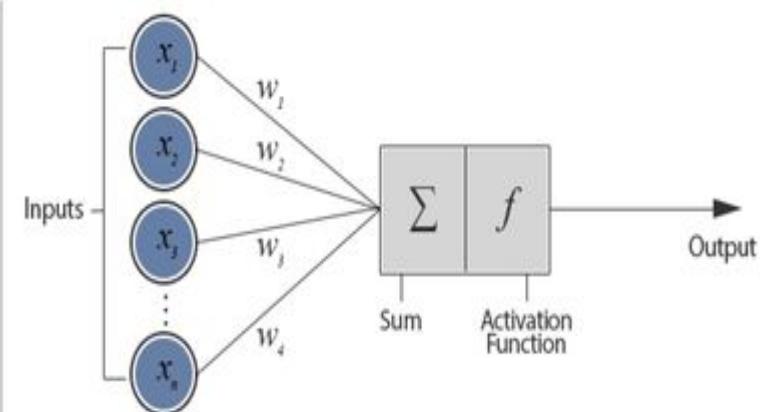
Идея имеет истоки в ранних работах физиологов, высказавших гипотезу о том, как может быть устроена работа мозга.

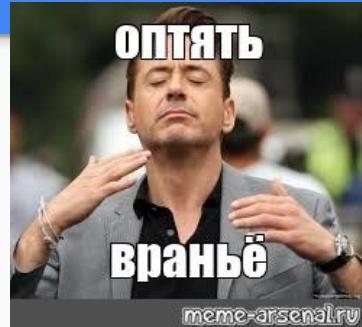


Предпосылки развития

Идея имеет истоки в ранних работах физиологов, высказавших гипотезу о том, как может быть устроена работа мозга.

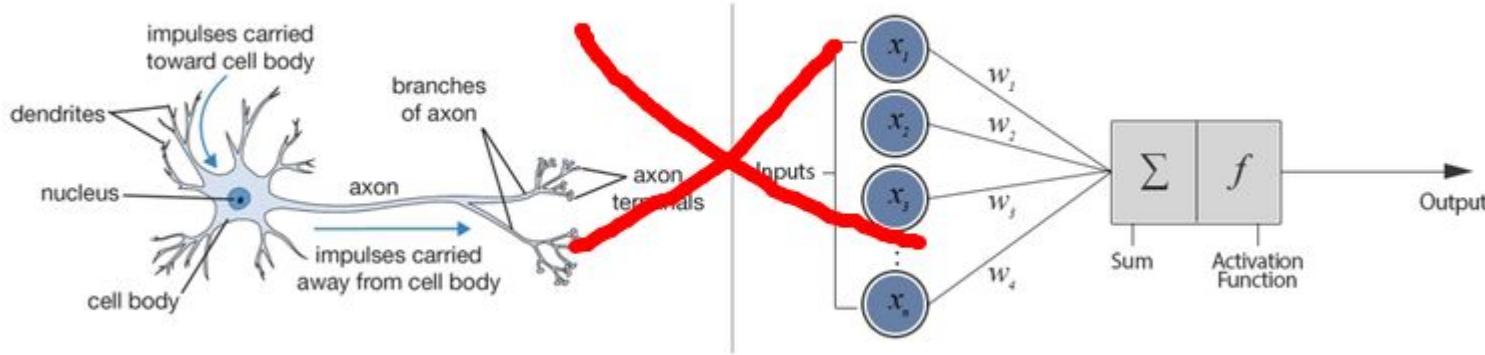
Следуя этой гипотезе, нейрон представляет собой нелинейную функцию от многих входов





Данная гипотеза не имеет ничего общего с действительностью!

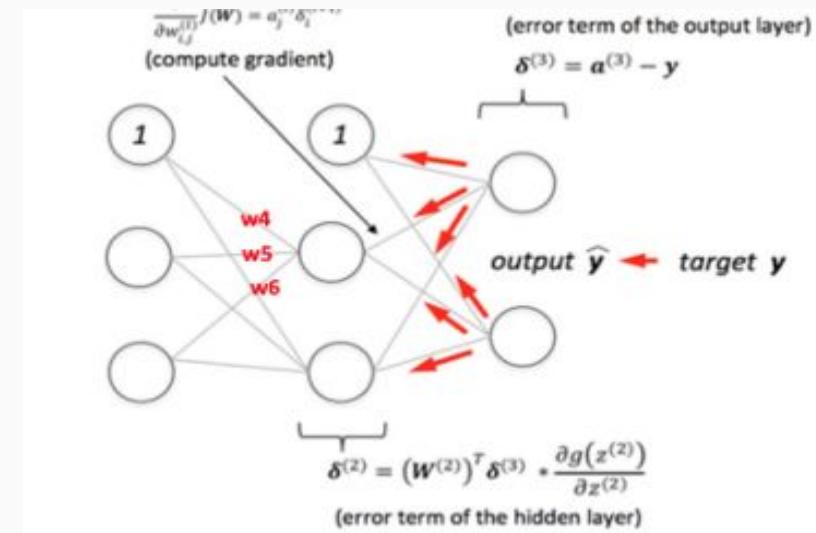
Biological Neuron versus Artificial Neural Network



Предпосылки развития

Back Propagation алгоритм

- David Rumelhart, +Geoff Hinton и другие в 80 -х годах 20 века.

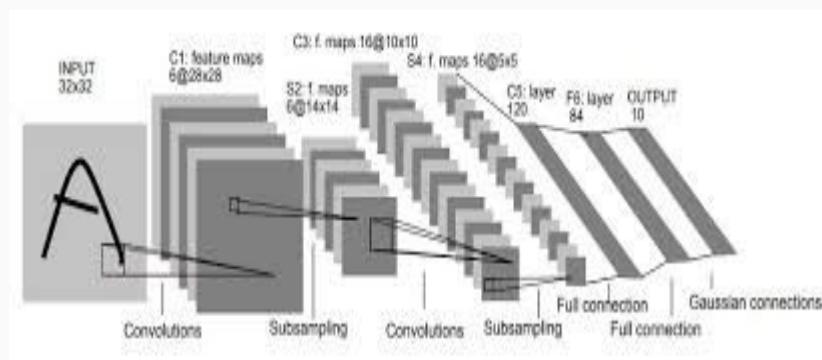


Предпосылки развития

Back Propagation алгоритм

Сверточные нейронные сети

- Yann LeCun - конец 90-х годов прошлого века
- (*The LeNet architecture was first introduced by LeCun et al. in their 1998 paper,*)



Предпосылки развития

Back Propagation алгоритм

Сверточные нейронные сети

Большие массивы данных



Предпосылки развития

Back Propagation алгоритм

Сверточные нейронные сети

Большие массивы данных

Дешевые вычисления



Предпосылки развития

Back Propagation алгоритм

Сверточные нейронные сети

Большие массивы данных

Дешевые вычисления

Работа большого числа ученых



Предпосылки развития

Back Propagation алгоритм

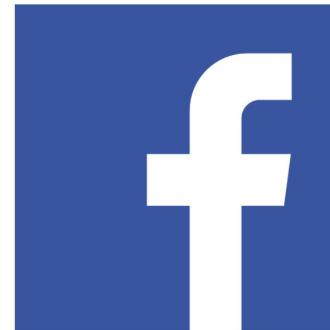
Сверточные нейронные сети

Большие массивы данных

Дешевые вычисления

Работа большого числа
ученых

Интерес больших корпораций



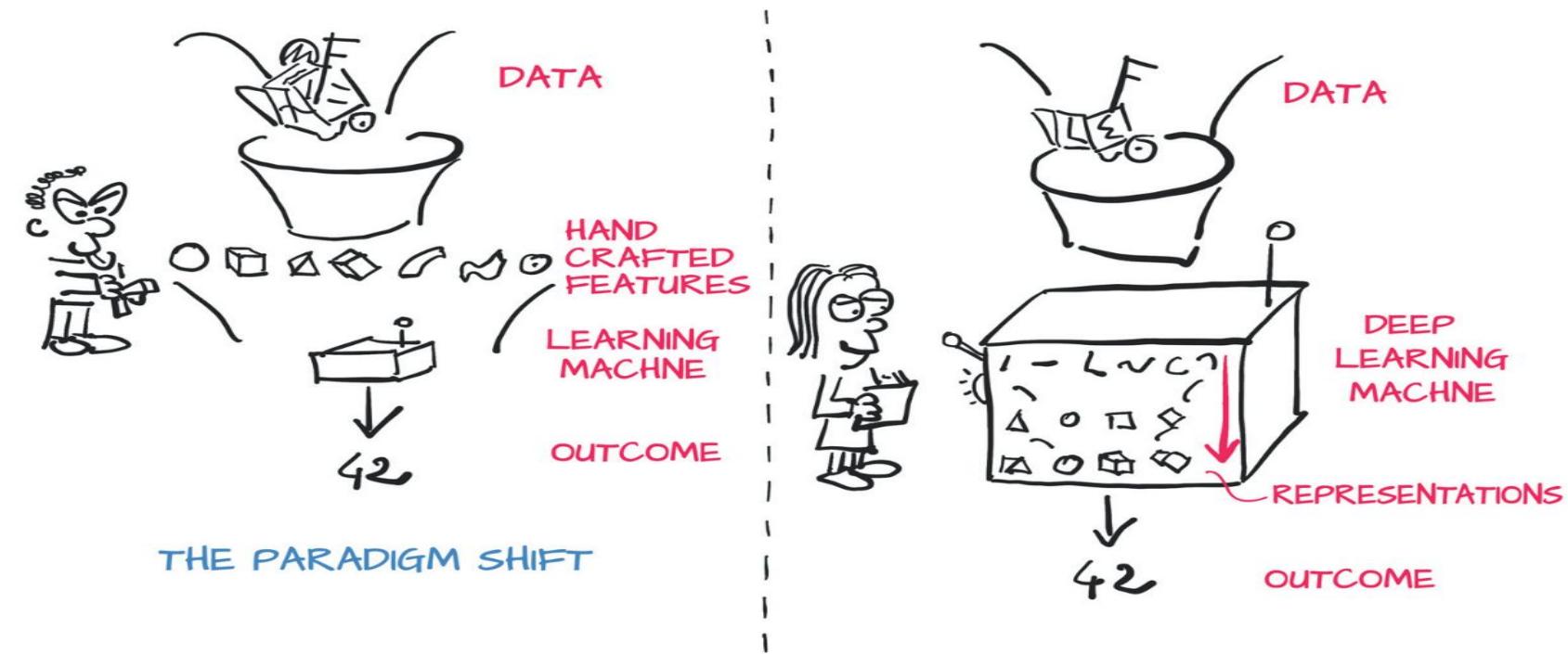
Helicopter View

Давайте рассмотрим предмет с высоты **птичьего полета**.

Наша цель - получить общее понимание (**intuition**) того, что из себя представляет глубокое обучение.



Deep Learning

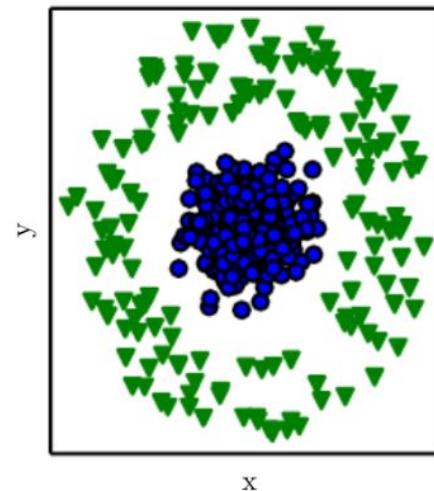


Deep Learning

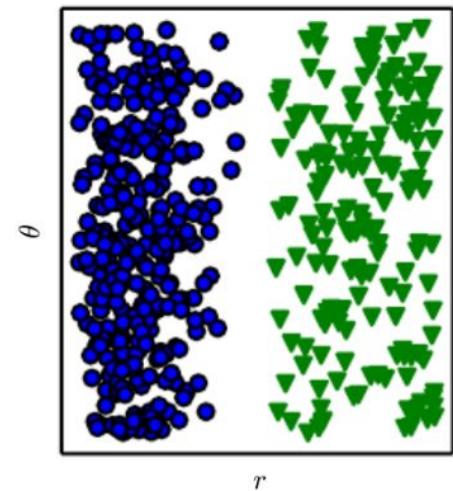
Решает проблему представления данных.

Основная проблема классического ML - представление данных в форме, позволяющее эффективно учить модель.

Cartesian coordinates

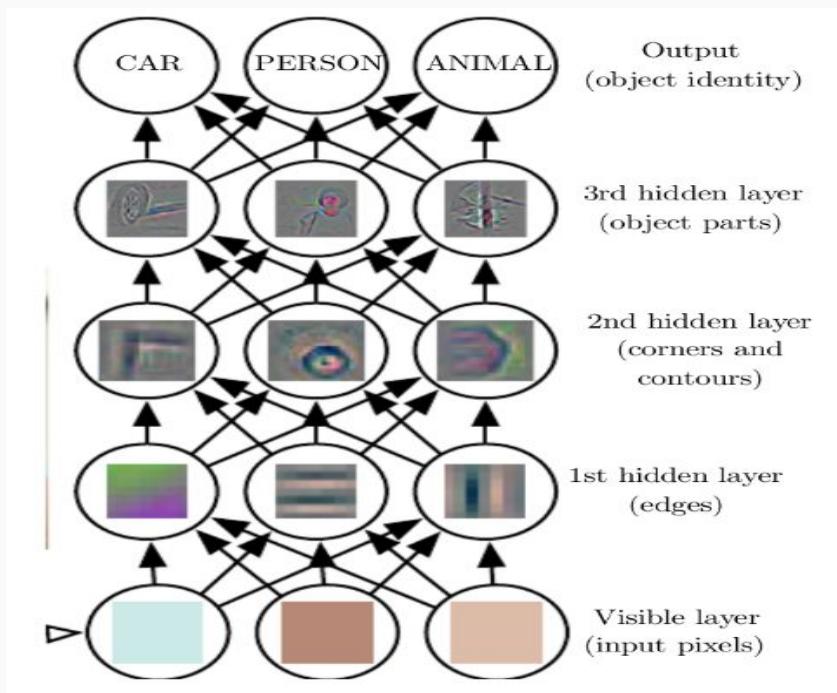
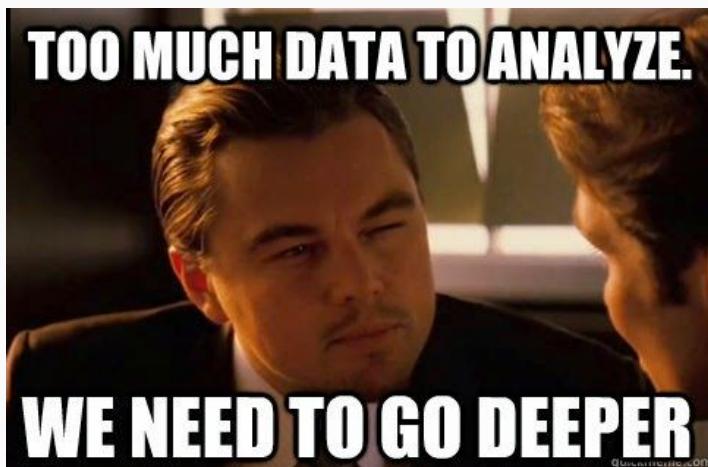


Polar coordinates



Deep Learning

DL решает проблему
эффективного представления
признаков путем построения
иерархии представлений.



Сфера применения (Несколько примеров)

- Обработка изображений (CV)
- Машинный перевод
- Генерация текстов
- Генерация изображений
- Детекция аномалий
- Задачи предсказания (Forecasting)
- Генерация изображений и видео



Отличия от классических методов машинного обучения

Позволяет избежать ручной экстракции признаков

Потребляет много ресурсов

Требует много данных для обучения (что правда только отчасти)

Наше все, так как успехи последних лет в обработке текстов, изображений и звука связаны именно с Deep Learning

Deep Learning

Что такое **искусственная нейронная сеть** (Artificial neural network)

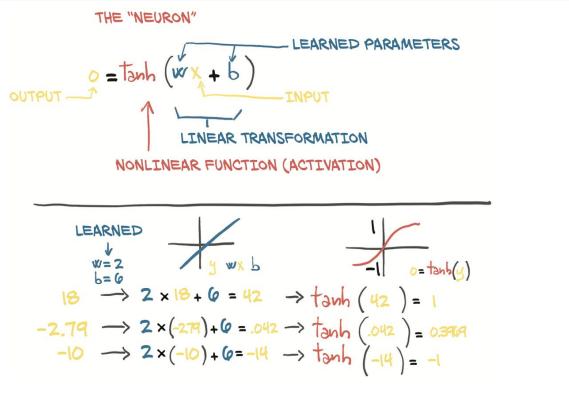
Вспомним постановку задачи обучения с учителем. У нас есть:

Обучающая выборка в виде набора X, Y

Некий набор функций, определяющий модель $f(X, W, b)$

Функция ошибки L , определяющая то, на сколько хорошо модель “умеет” воспроизводить отображение из X в Y

Некая процедура, позволяющая настроить веса модели так, чтобы величина ошибки была минимальной на наборе X, Y



Что такое **искусственная нейронная сеть** (Artificial neural network)

Тогда такая модель называется искусственной нейронной сетью, если она имеет вид:

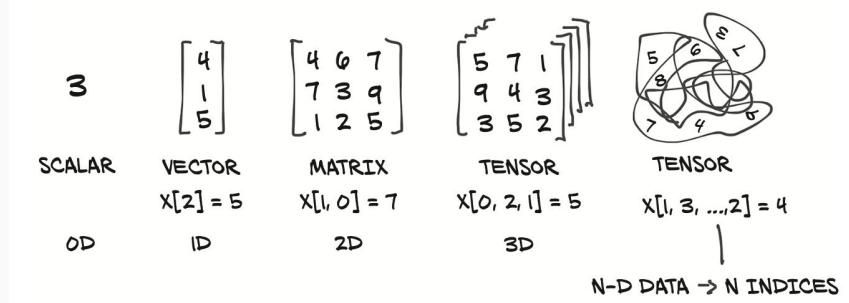
$$\begin{aligned}\mathbf{h}^{(1)} &= g^{(1)}(\mathbf{W}^{(1)\top} \mathbf{x} + \mathbf{b}^{(1)}) \\ \mathbf{h}^{(2)} &= g^{(2)}(\mathbf{W}^{(2)\top} \mathbf{h}^{(1)} + \mathbf{b}^{(2)}) \\ \dots \\ \mathbf{h}^{(n)} &= g^{(n)}(\mathbf{W}^{(n)\top} \mathbf{h}^{(n)} + \mathbf{b}^{(n)})\end{aligned}$$

Где $\mathbf{W}_i \sim U(-\frac{1}{\sqrt{m}}, \frac{1}{\sqrt{m}})$

\mathbf{b}_i - Const, $0 < b_i < 1$

Замечание

$\mathbf{W}_i, \mathbf{h}_i, \mathbf{x}_i, \mathbf{b}_i$ - могут быть **тензорами** произвольной размерности.



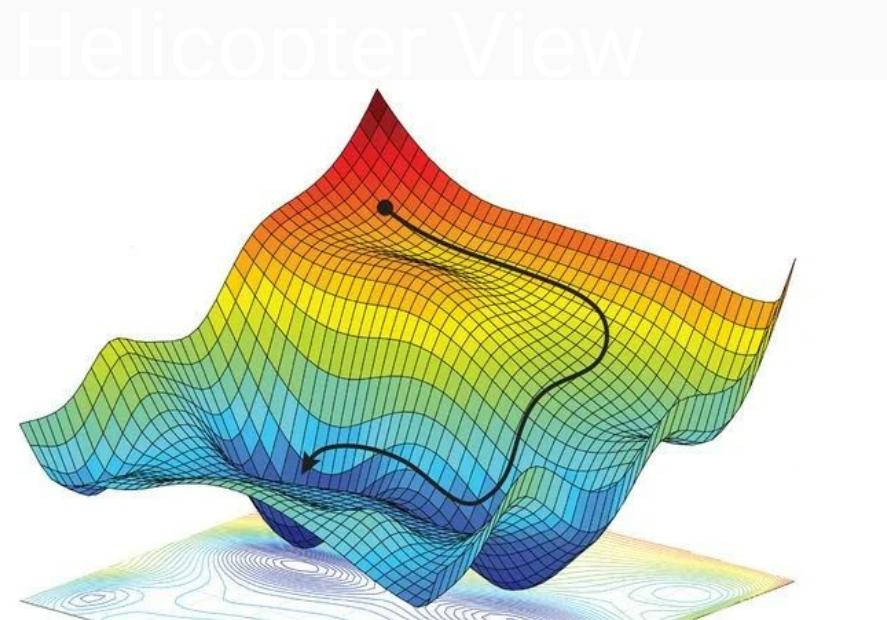
Как обучаются искусственные нейронные сети ?

Для обучения используется все тот же метод **градиентного спуска** и его модификации.

$\min(L(F(X,Y,W,b)))$ на множестве $\{X,Y\}$

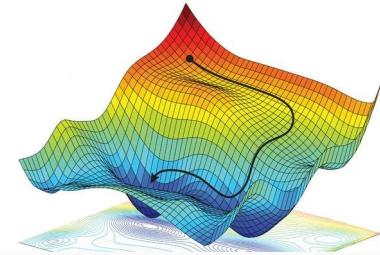
Основная тонкость заключается в том, что в случае искусственных нейронных сетей необходимо производить дифференцирование большого числа многомерных объектов (**тензоров**).

Для того чтобы делать это эффективно, была изобретена так называемая процедура **обратного распространения ошибки**.



Метод градиентного спуска (Метод градиентного спуска)

Helicopter V



Require: Learning rate ϵ_k .

Require: Initial parameter θ

while stopping criterion not met **do**

 Sample a minibatch of m examples from the training set $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$ with corresponding targets $\mathbf{y}^{(i)}$.

 Compute gradient estimate: $\hat{\mathbf{g}} \leftarrow +\frac{1}{m} \nabla_{\theta} \sum_i L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$

 Apply update: $\theta \leftarrow \theta - \epsilon \hat{\mathbf{g}}$

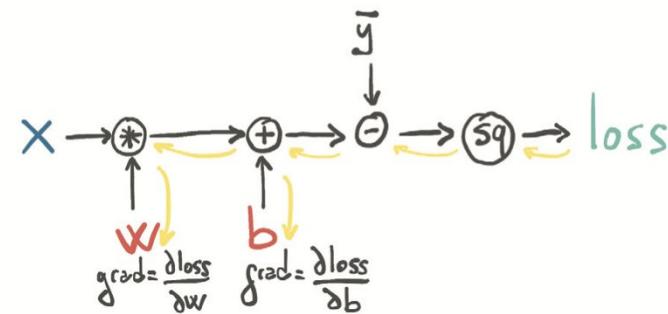
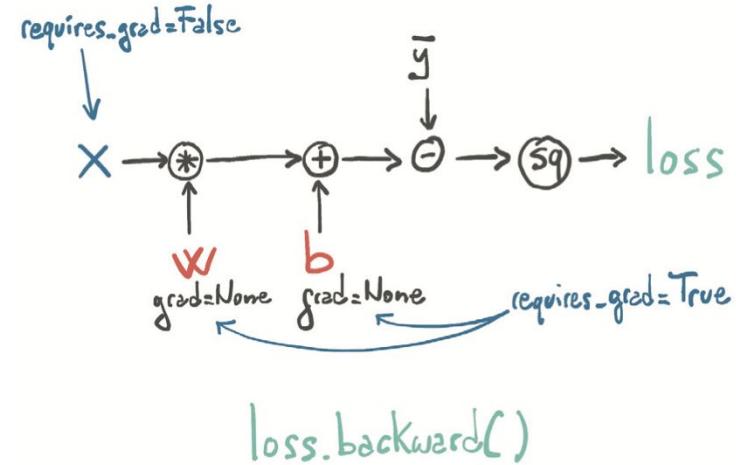
end while

Deep Learning

Как обучаются искусственные нейронные сети?

Основные понятия, про которые нужно знать

- Граф вычислений
- Chain rule
- Back propagation
- Batch, mini batch



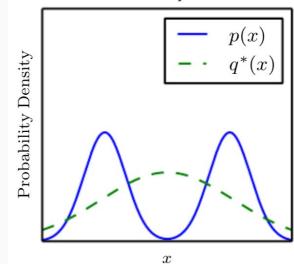
Функция ошибки (Cost function)

В большинстве случаев модель определяет распределение

$$p(y \mid x; \theta)$$

и мы используем принцип **максимального правдоподобия**, что значит мы используем **кросс энтропию** между предсказаниями модели и данными:

$$H(p, q) = - \sum_{x \in \mathcal{X}} p(x) \log q(x)$$



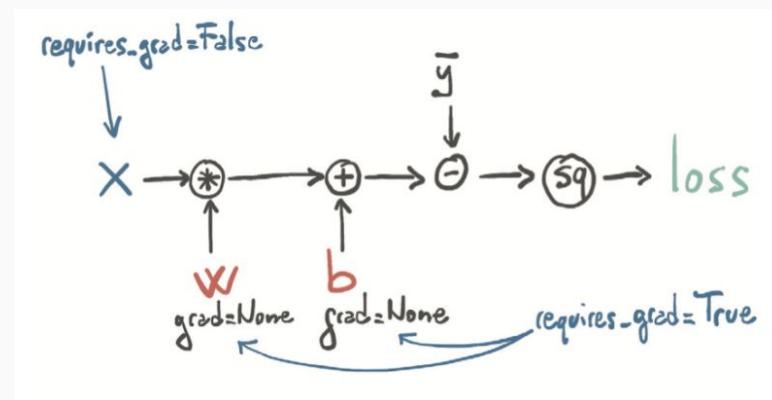
Из чего состоит процесс обучения ?

- Разбиение обучающей выборки на три части: Train, Test, Validation
- Разбиение Train на части меньшего размера (Mini Batch) (8, 16 32, 64, 128 ...)
- Запуск процесса обучения на каждом из полученных (Mini Batch)
- Проверка полученных результатов на Validation выборке
- В конце процесса запускаем проверку полученных результатов на Test выборке

Deep feedforward networks

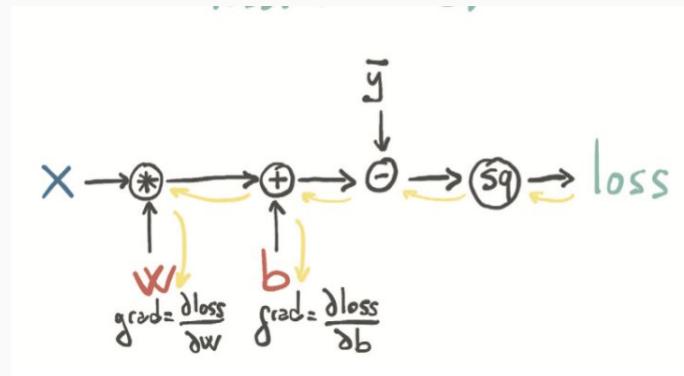
Цикл обучения состоит из трех фаз:

- Forward propagation (Прямое распространение)



Цикл обучения состоит из трех фаз:

- Back propagation (Обратное распространение)



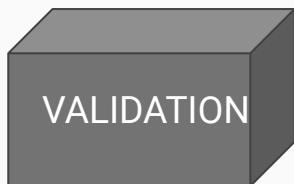
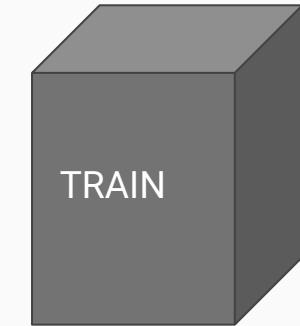
Цикл обучения состоит из трех фаз:

- Обновление весов

$$W = W + \text{grad}_W(L)$$

$$b = b + \text{grad}_b(L)$$

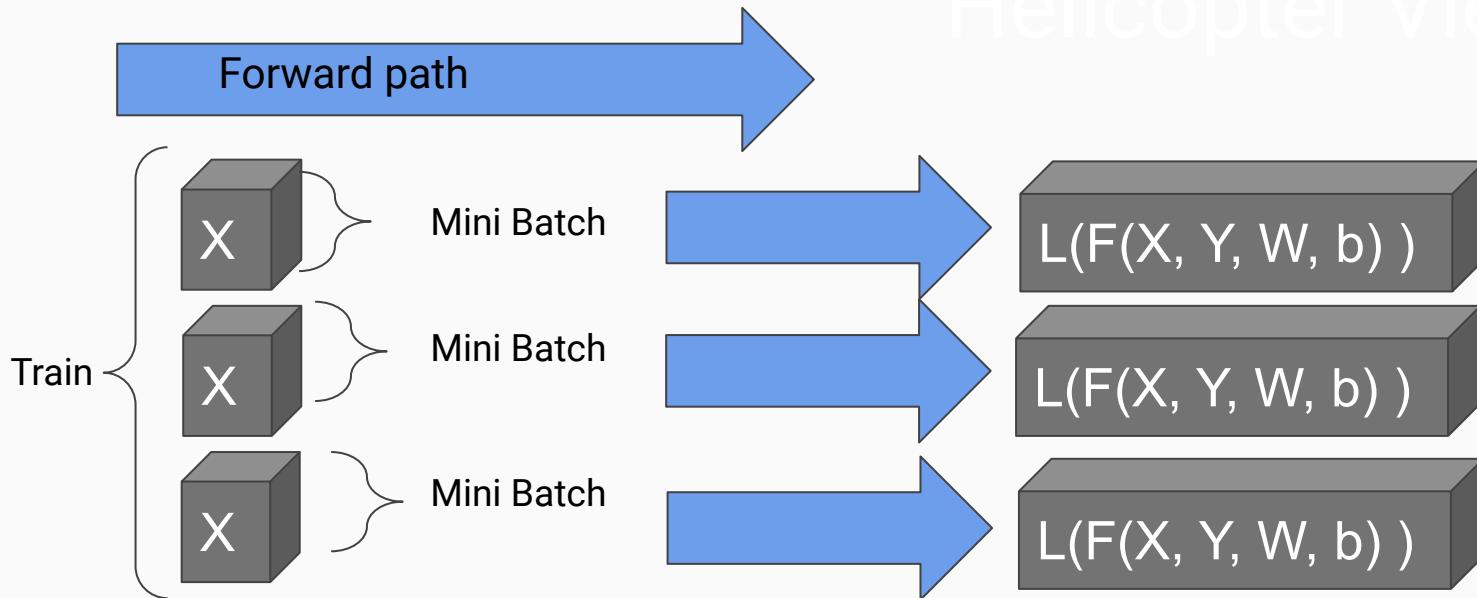
Разбиваем данные



Так как данных как правило много, разбивают как правило в соотношении 98 : 1 : 1

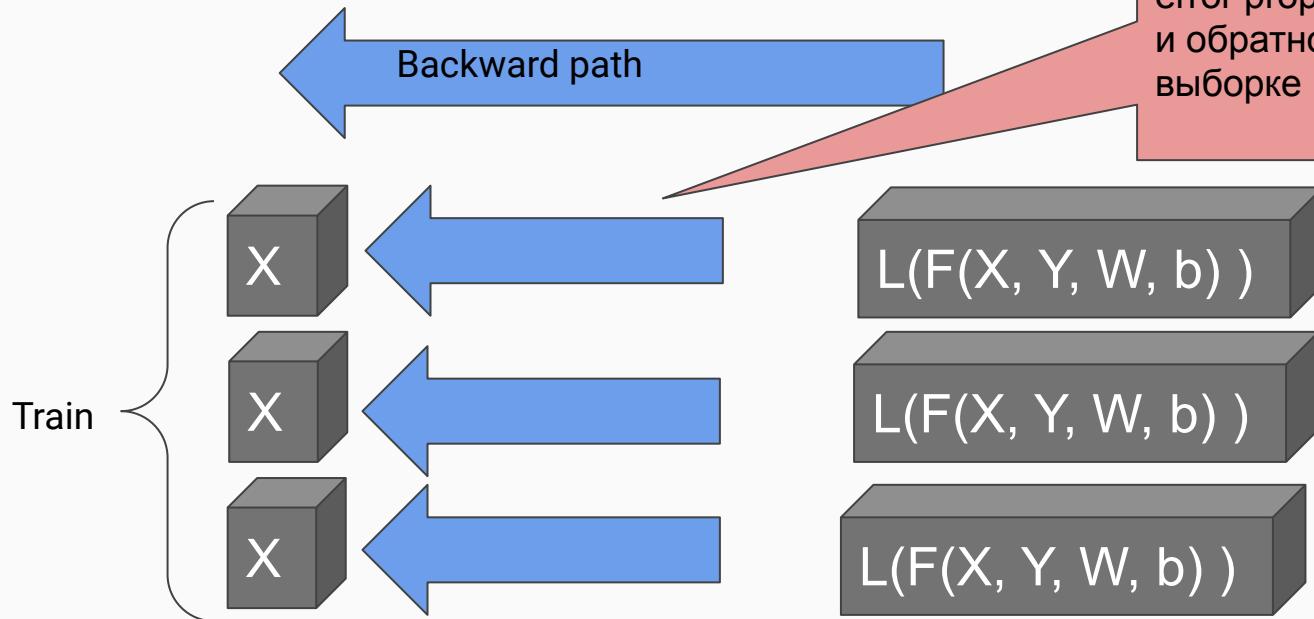
Deep Learning

Вычисляем функцию ошибки на каждом из батчей



Deep Learning

Вычисляем градиент ошибки относительно весов и обновляем веса

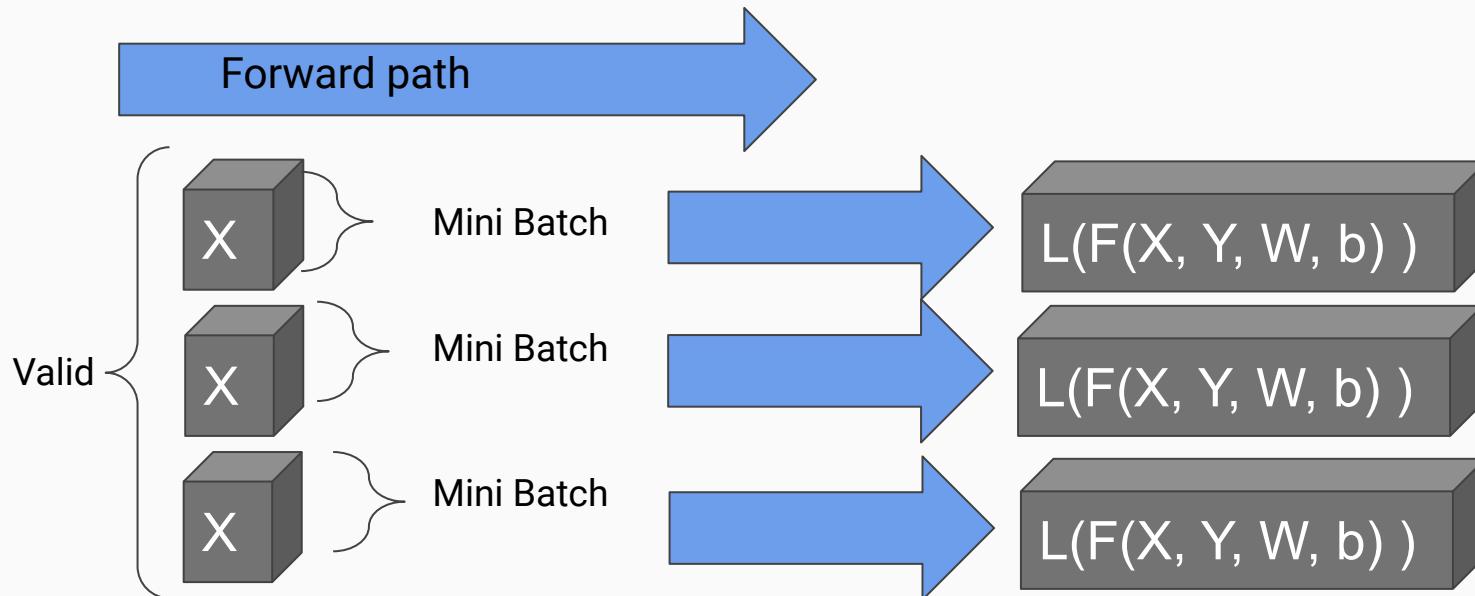


Градиент весов вычисляется “Задом наперед” От этого по название “Back error propagation” Один цикл прямого и обратного прохода по Training выборке называется эпохой



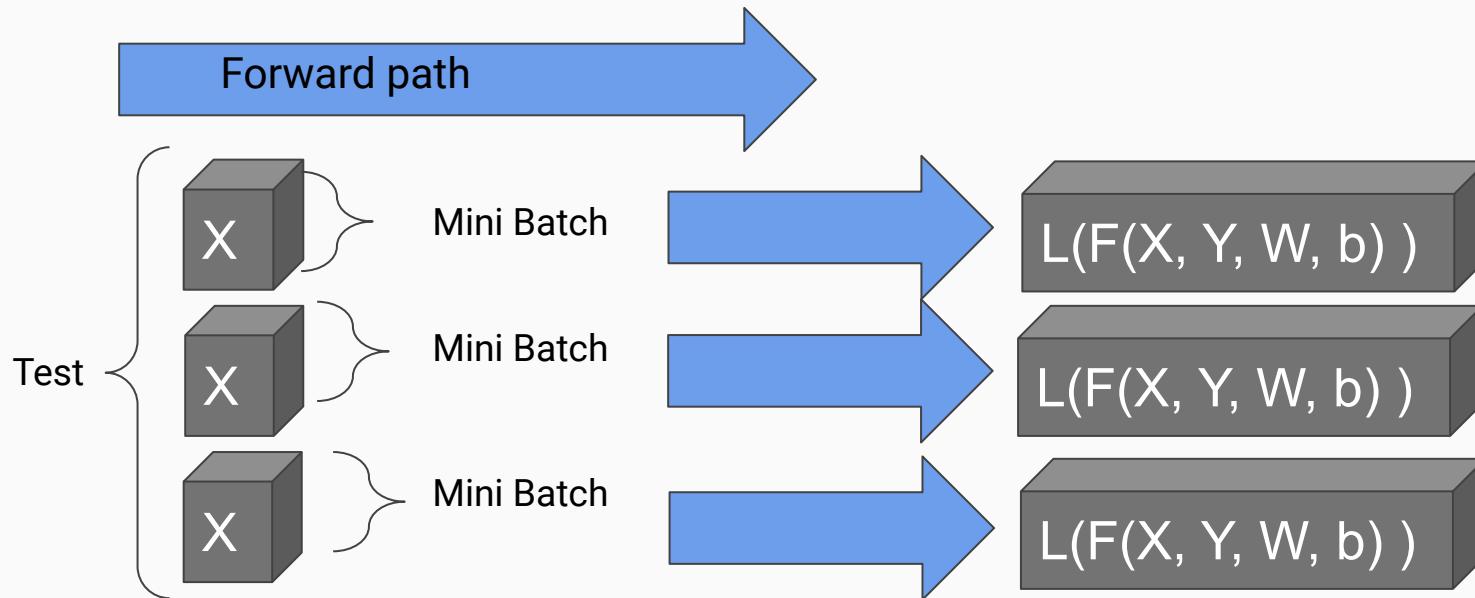
Deep Learning

В конце каждой эпохи вычисляем функцию ошибки (Либо какую то другую интересующую нас метрику) на валидационной выборке



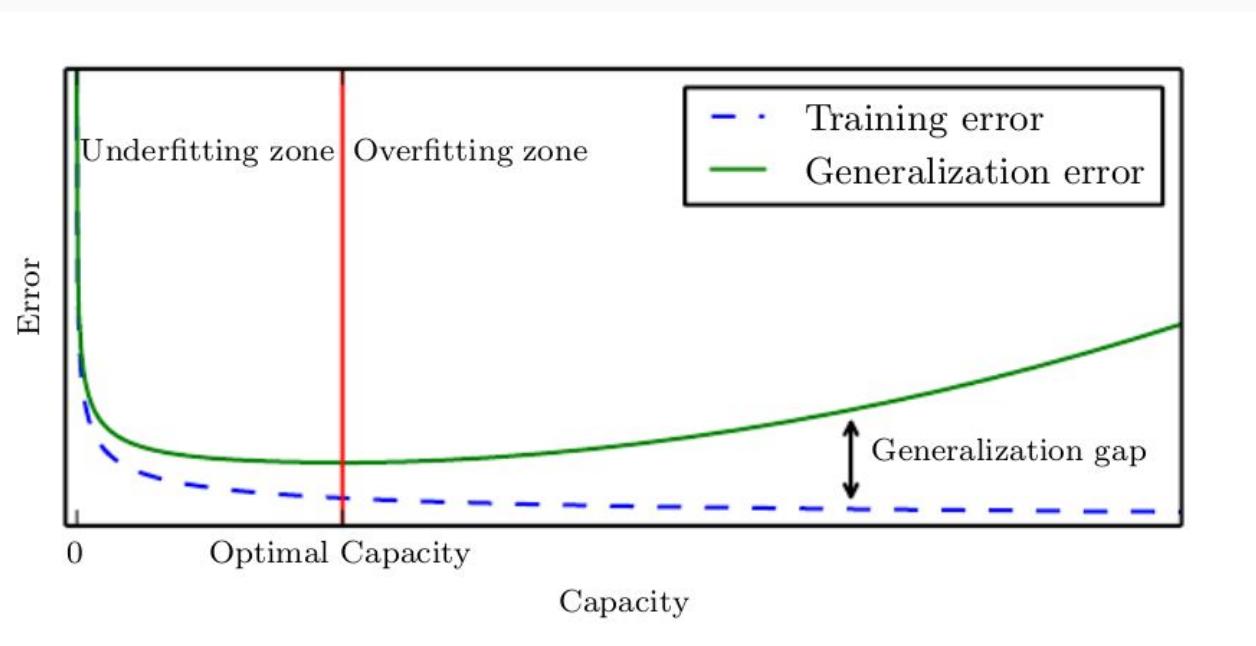
Deep Learning

По завершению процесса обучения вычисляем функцию ошибки (Либо другую интересующую нас метрику) на тестовой выборке



Deep Learning

Чего мы хотим добиться ?

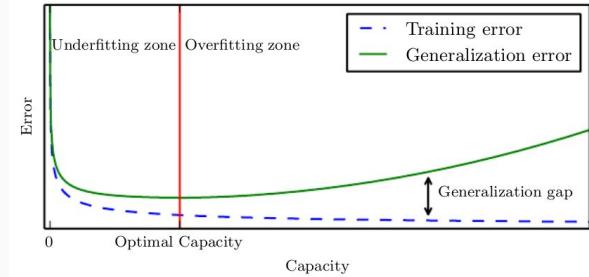


Основная цель - уменьшить ошибку на обучающей выборке не увеличивая на тестовой

Deep Learning

Здесь у нас могут возникнуть следующие ситуации:

1. Переобучение (Overfitting) - модель переобучивается на training множестве но на validation показывает плохой результат
2. Недообучение (Underfitting) - Модель показывает плохой результат на training множестве.
3. Just fine - Модель показывает хорошие результаты и на train и на Validation



ЧТО НУЖНО ДЕЛАТЬ В КАЖДОМ ИЗ СЛУЧАЕВ ?
Давайте обсудим!

Архитектура (Architecture design)

Что интересует ?

Точность (**Presicion**) - мы хотим иметь по возможности высокую точность

Скорость (**Latency**) - мы хотим иметь по возможности низкую латентность

Размер моделей - особо актуально для мобильных приложений

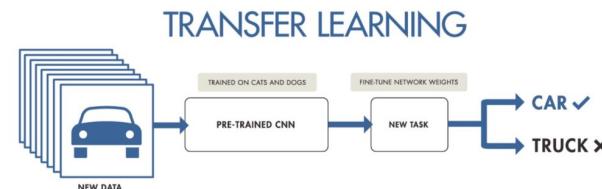
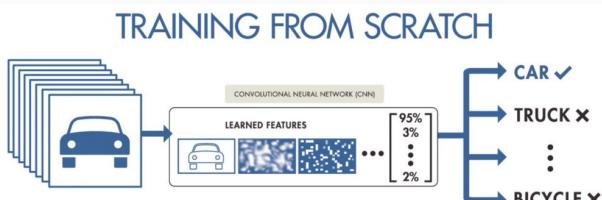
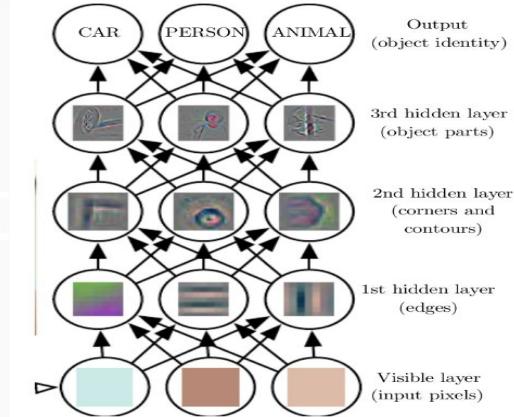
Deep Learning

Перенос знаний или **transfer learning**

Идея **transfer learning** состоит в том, что мы можем использовать ранее натренированные веса сетей для решения своих задач

Таким образом, обучив однажды сеть на большом объеме данных для решения какой либо задачи, можно использовать эту же сеть с минимальными модификациями для решения других.

Это свойство является одним из самых **важных** для нас как практиков, так как позволяет нам обучать уже обученные сети на относительно небольших объемах данных.



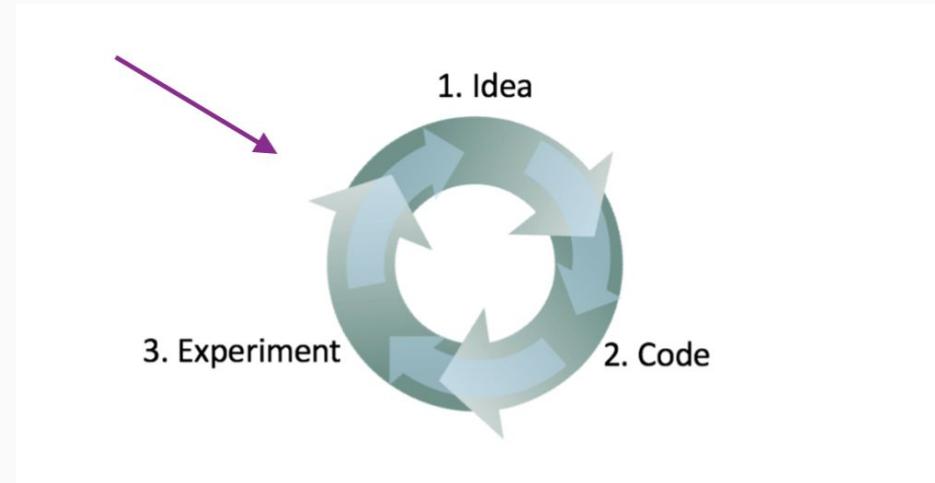


Практическая методология

1. Процесс является **итерационным и не ложится на водопадную модель**
2. Процесс является **эмпирическим**

Поэтому для достижения успеха:

- Сформулировать цель
- Собрать набор данных
- Как можно раньше начать эксперименты
- Двигаться итерационно



Сверточные сети в двух словах

Вспомним как выглядит классическая нейронная сеть:

$$\begin{aligned}\mathbf{h}^{(1)} &= g^{(1)}(\mathbf{W}^{(1)\top} \mathbf{x} + \mathbf{b}^{(1)}) \\ \mathbf{h}^{(2)} &= g^{(2)}(\mathbf{W}^{(2)\top} \mathbf{h}^{(1)} + \mathbf{b}^{(2)}) \\ \dots \\ \mathbf{h}^{(n)} &= g^{(n)}(\mathbf{W}^{(n)\top} \mathbf{h}^{(n)} + \mathbf{b}^{(n)})\end{aligned}$$

Основные преимущества сверточных сетей над полносвязными:

- Устойчивость к сдвигу
- Меньший объем

Если заменим операцию умножения на операцию свертки, такая сеть называется **сверточной**

Сверточные сети

Свертка (**convolution**) функций x и w представляется так:

$$s(t) = \int x(a)w(t-a)da.$$

Или в другом виде :

$$s(t) = (x * w)(t).$$

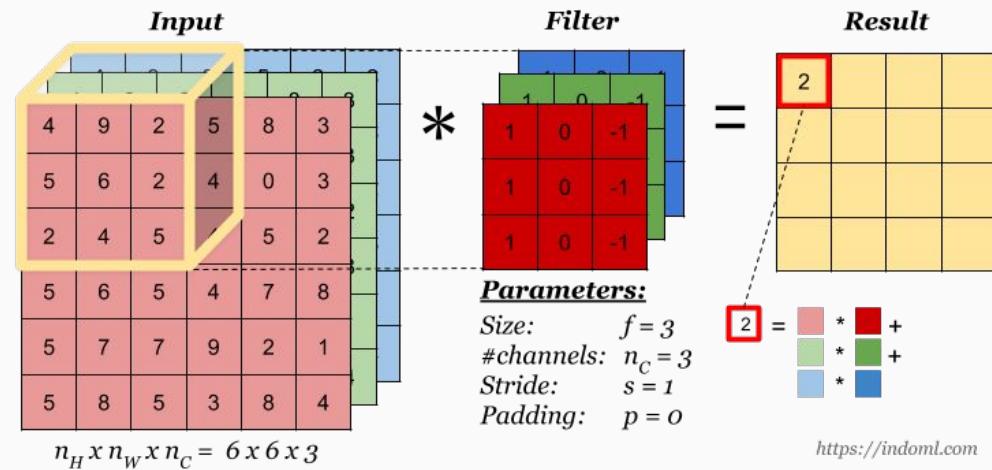
В этом случае **x** называют **входом**, а **w** - **ядром**.

Сверточные сети

Операция свертки

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(m, n)K(i - m, j - n).$$

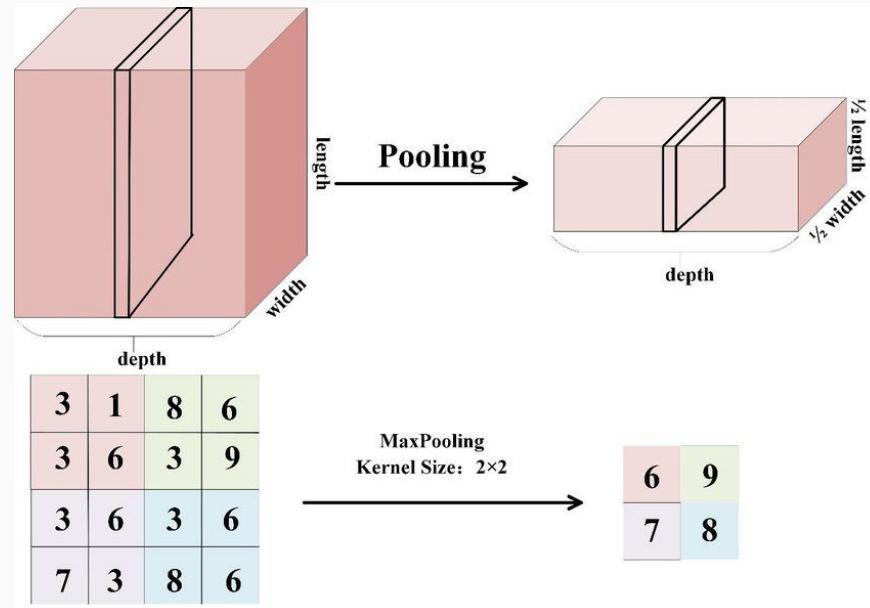
Сверточные сети Операция свертки



Сверточные сети

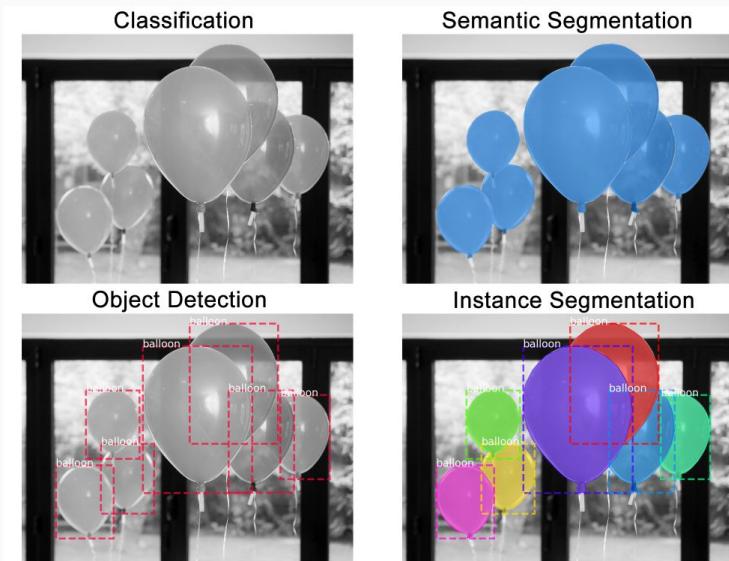
Операция пулинга (pooling)

Здесь важно понять то, что в отличии операции свертки, операция пулинга применяется к каждому из слоев отдельно



Deep Learning применительно к задачам CV

Задачи компьютерного зрения (Computer vision)



Классификация (Classification) - Определить есть ли шарик на данном рисунке.

Семантическая сегментация (Semantic segmentation)
- Найти все пиксели, относящиеся к шарику.

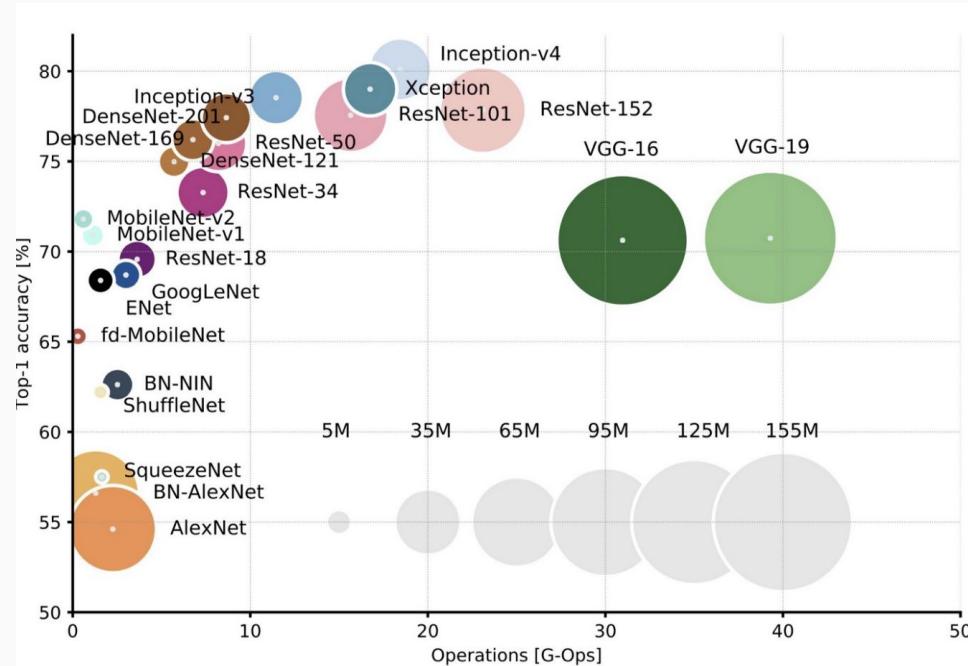
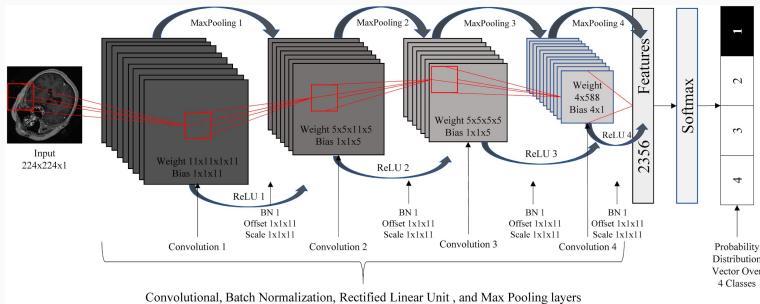
Детекция объекта (Object detection) Выделить все регионы, на которых присутствует шарик.

Instance Segmentation Выделить все регионы, на которых присутствует шарик, при этом выделив все пиксели, относящиеся к каждому из объектов.

Deep Learning применительно к задачам CV

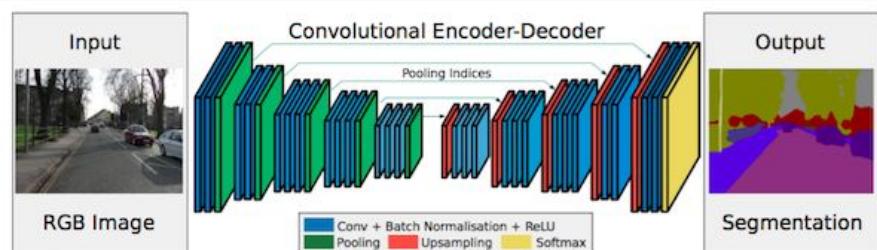
Классификация (Classification)

Применяются различные виды сверточных сетей Convolution Neuron Networks или CNN



Семантическая сегментация (**Semantic segmentation**)

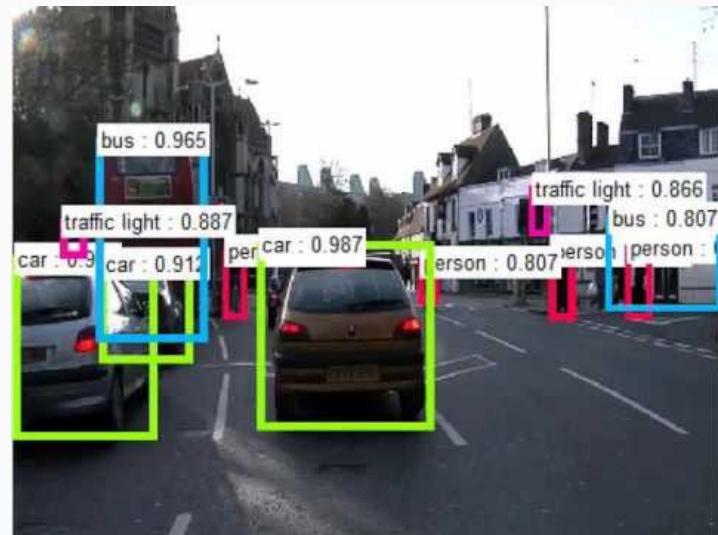
Применяются различные виды сверточных сетей Fully convolution networks или FCN таких как:
U-Net



Детекция (Object detection)

Применяются различные виды алгоритмов, таких как
YOLO-v1/2/3
Fast-R-CNN
Faster-R-CNN

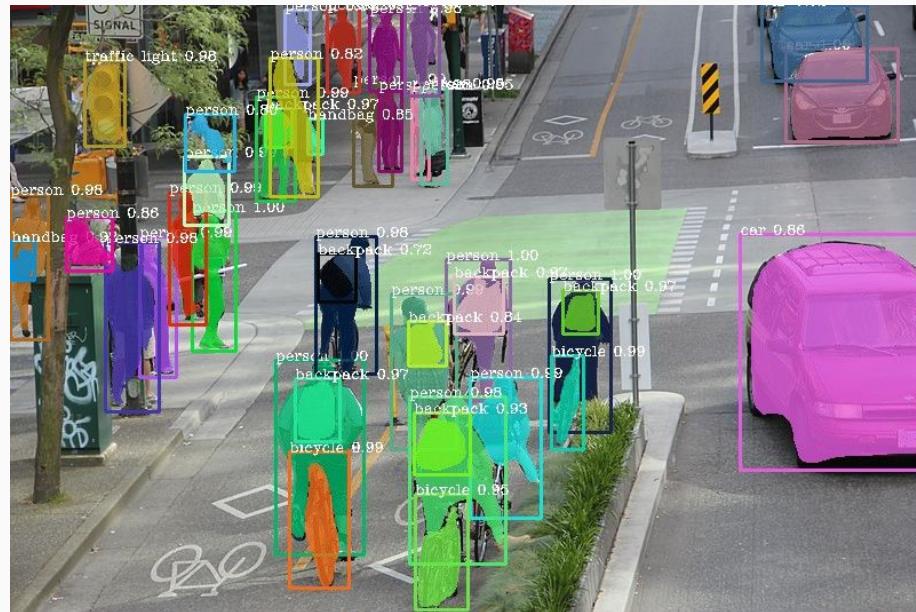
RCNN - Region-based Convolutional Network



Deep Learning применительно к задачам CV

Instance Segmentation

Применяются различные виды алгоритмов, таких как Mask-RCNN и ее дальнейшие разработки



Self-supervised learning

Проблема! Разметка данных стоит
дорого! Что делать?

Идея! : Разбить процесс обучения на
два этапа:

- Обучить сеть на большой
неразмеченной выборке
используя некую промежуточную
меру сходства.
- Дообучить полученную сеть на
небольшой **размеченной**
выборке



Self-supervised learning

Отличие **self-supervised learning** от **supervised learning** состоит в том, что у нас нет размеченных данных

supervised learning



CAT

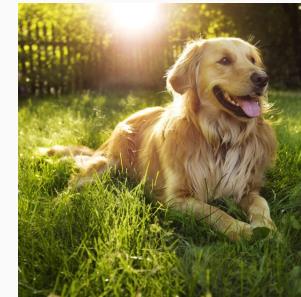


DOG

self-supervised learning



???



???

Self-supervised learning

На сегодняшний день существует несколько таких **self-supervised** алгоритмов обучения

Мы коротко рассмотрим один из самых простых из них - **SimCLR**

На входе мы имеем два набора
данных:

- Большой неразмеченный
- Маленький размеченный

В этом случае обучение производится в два этапа:

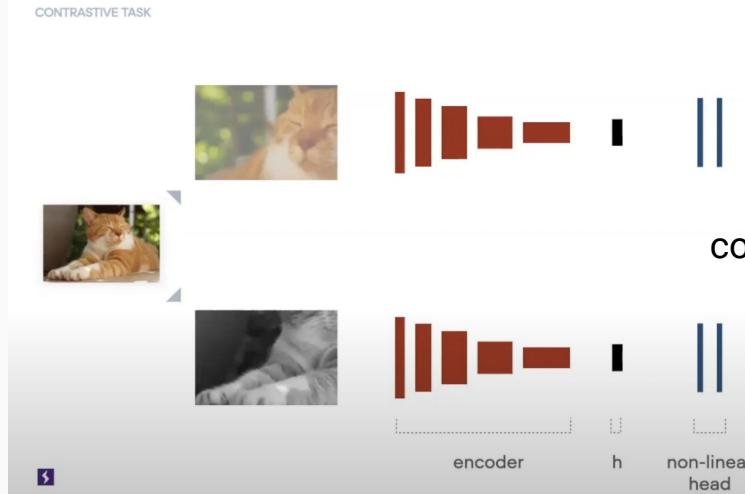
На первом этапе модель обучают на большом неразмеченном наборе, используя функцию ошибки определенного вида contrastive-loss .

На втором этапе часть слоев модели выбрасывают, а часть до обучают на небольшом размеченном наборе данных

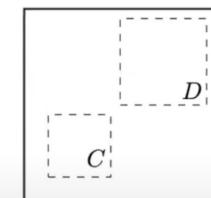
Self-supervised learning

На сегодняшний день существует несколько таких self-supervised алгоритмов обучения

Мы рассмотрим самый простой из них - **SimCLR**



Sim CLR:



(b) Adjacent views.

Self-supervised learning

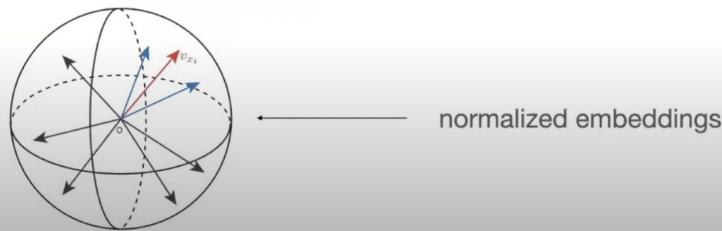
contrastive loss

$$\text{sim}(\mathbf{u}, \mathbf{v}) = \mathbf{u}^\top \mathbf{v} / \|\mathbf{u}\| \|\mathbf{v}\|$$

Temperature and normalization

$$l_{i,j} = -\log \frac{\exp(\text{sim}(z_i, z_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(\text{sim}(z_i, z_k)/\tau)}$$

that tau's imp.



OVERVIEW

Go big or go home!



VS



larger batch!
more negatives!
longer training time!



Self-supervised learning

Для дальнейшего изучения вопроса читайте оригиналную статью:

<https://arxiv.org/pdf/2002.05709.pdf>

Также есть описание реализации алгоритма от авторов pytorch-lightning

https://www.youtube.com/watch?v=7QmsTleiRLs&list=PLaMu-SDt_RB4k8VXiB3h0dsn0Y3GoXo1k

Как мы применяем DL и CV в RGS

- Определение комплектности предстрахового осмотра.
- Определение степени загрязнения авто
- Определение степени повреждения авто
- Выделение части авто и для каждой из частей определить степень повреждения

Определение комплектности осмотра

Постановка задачи - нам нужно определить, все ли необходимые фото нужного качества присутствуют в комплекте фото, сделанных на этапе предстрахового осмотра.

Осмотр считаем комплектным, если на нем присутствуют:

- Фото агента на фоне авто
- Авто, снятое с пяти ракурсов
- Приборная панель
- Колесо крупным планом
- VIN номер
- Ветровое стекло

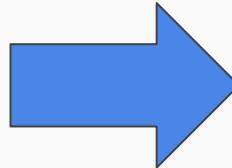
Таким образом, перед нами задача **классификации**

Эту задачу решаем в два этапа

Как мы применяем DL и CV в RGS

Определение комплектности осмотра

- Этап 1. Определение того, к какому классу принадлежит снимок



AGENT

CAR

DASHBOARD

OTHER

VIN

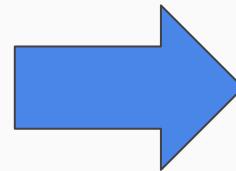
WHEEL

WINDSHIELD

Как мы применяем DL и CV в RGS

Определение комплектности осмотра

- Этап 2. Определяем угол, под которым сделан снимок авто, если на предыдущем этапе было авто

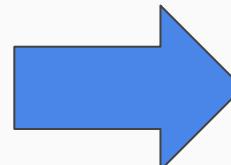


BACK_LEFT
BACK_RIGHT
FRONT_LEFT
FRONT
FRONT_RIGHT

Как мы применяем DL и CV в RGS

Определение комплектности осмотра

- Этап 3. Определяем дополнительные характеристики - в данном случае признак загрязненности



CLEAN

DIRTY

Как мы применяем DL и CV в RGS

Определение комплектности осмотра

Как результат, на выходе представленного алгоритма получаем ряд метрик, таких как признак полноты осмотра и загрязненности авто, что позволяет оценить степень пригодности последнего

Определение степени повреждения

- **Постановка задачи:** Нам необходимо определить степень повреждения авто, имея фото этого авто, сделанных с разных ракурсов.

Как мы применяем DL и CV в RGS

Определение степени повреждения

Выделение деталей



Как мы применяем DL и CV в RGS

Определение степени повреждения

Выделение повреждений



Как мы применяем DL и CV в RGS

Определение степени повреждения

Расчет степени повреждения (Берется пересечение области делали и повреждения)



5% процентов бампера повреждено. Тип повреждения scratch



5% процентов бампера повреждено. Тип повреждения crack



20% процентов двери повреждено. Тип повреждения scratch

P Y T H O N

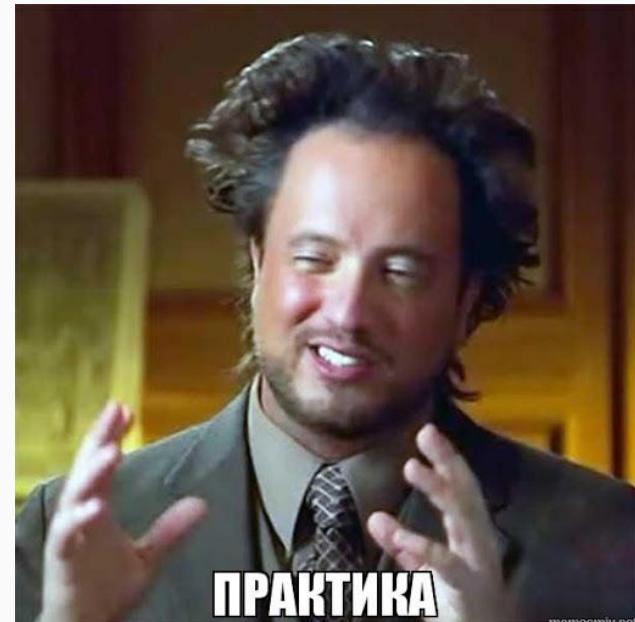


PyTorch
Lightning

Практика

Наша задача - познакомится с тем, как на практике решаются задачи глубокого обучения.

На сегодняшний день существует множество фреймворков и библиотек, реализующих подходы глубокого обучения. В этой лекции мы начнем изучать один из таких фреймворков.

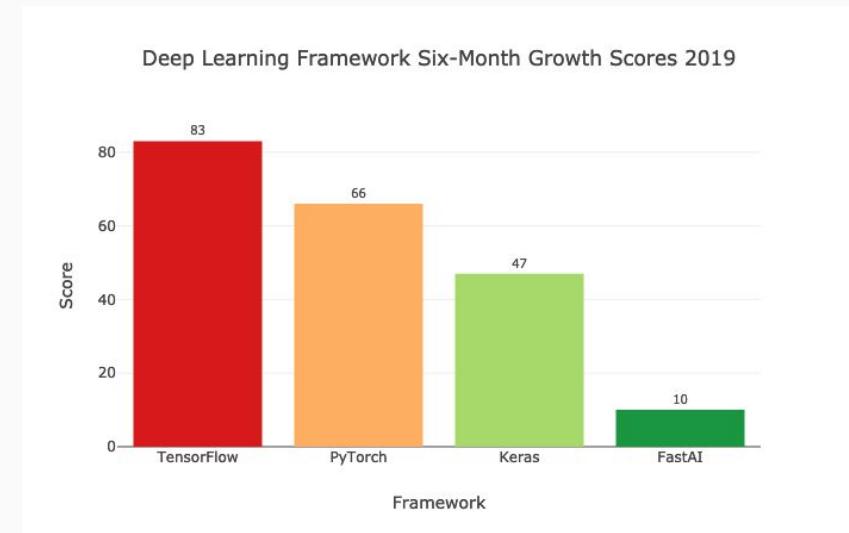




Практика

Два наиболее популярных на сегодняшний день фреймворка:

- Tensorflow
- Pytorch





Tensorflow

Pros:

- Есть собственные механизмы построения pipeline
- Большое комьюнити

Cons:

- Код тяжело читать так как он не `not Python like style`

Pytorch

Pros:

- Легок в изучении
- Концептуально чист
- Популярен среди ресечеров
- Популярен в российской тусовке
- Набирает все большую популярность

Cons:

- Не обеспечивает построения pipeline из коробки

PyTorch : Уровни абстракции:

Tensor

Представляет собой ndarray с возможностью вычислений на GPU

Variable

Нода в графе вычислений. Хранит результаты вычислений

Module

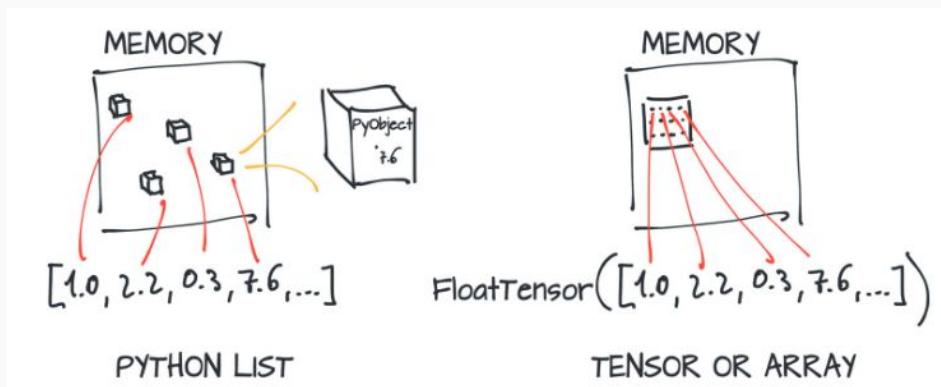
Слой нейронной сети

Optimiser

Алгоритмы оптимизации

PyTorch :Tensor

Многомерный , компактно хранимый массив.



PyTorch :Tensor

Существует множество способов инициализации

Вот некоторые из них:

```
t1 = torch.ones(10)
print('t1 :', t1)
t2 = torch.zeros(2,4)
print('t2 :', t2)
t3 = torch.tensor([[1, 2, 3],[3, 3, 6],[6, 7, 8]])
print('t3 :', t3)
t4 = torch.rand(3,3)
print('t4 :', t4)

t1 : tensor([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
t2 : tensor([[0., 0., 0., 0.],
           [0., 0., 0., 0.]])
t3 : tensor([[1, 2, 3],
           [3, 3, 6],
           [6, 7, 8]])
t4 : tensor([[0.2369, 0.9066, 0.0011],
           [0.8357, 0.8680, 0.4084],
           [0.1419, 0.7869, 0.0074]])
```

PyTorch :Tensor:Механизм хранения данных

С понятием тензора связано понятие **storage** или **хранилище**

Storage представляет собой плоский массив, хранящий данные нужного типа. (Например float или int32)

Основным свойством storage является то, что оно обеспечивает механизм компактного хранения данных в памяти.

Таким образом, всякий тензор является отображением его storage

PyTorch :Tensor:Механизм хранения данных

С понятием тензора связаны следующие понятия:

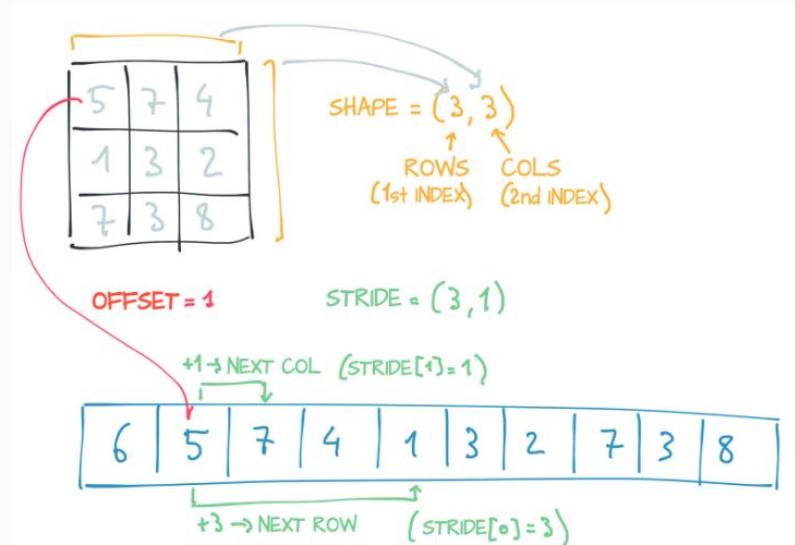
- **shape** - размерность по каждой из осей
- **offset** - индекс, указывающий адрес первого элемента тензора в хранилище
- **stride** - количество элементов, которые нужно пропустить, для того чтобы получить следующий элемент из каждого измерения.

PyTorch :Tensor:Механизм хранения данных

Из всего вышесказанного следует, что например доступ к элементу 2D тензора с координатами i, j можно получить по формуле:

storage_offset + stride[0] * i + stride[1] * j

Такой подход помимо всего прочего позволяет производить некоторые операции очень быстро, так меняется только представление данных.



PyTorch :Tensor

Типы данных

- `torch.float32` or `torch.float`—32-bit floating-point
- `torch.float64` or `torch.double`—64-bit, double-precision floating-point
- `torch.float16` or `torch.half`—16-bit, half-precision floating-point
- `torch.int8`—Signed 8-bit integers
- `torch.uint8`—Unsigned 8-bit integers
- `torch.int16` or `torch.short`—Signed 16-bit integers
- `torch.int32` or `torch.int`—Signed 32-bit integers
- `torch.int64` or `torch.long`—Signed 64-bit integers

PyTorch :Tensor

Типы данных

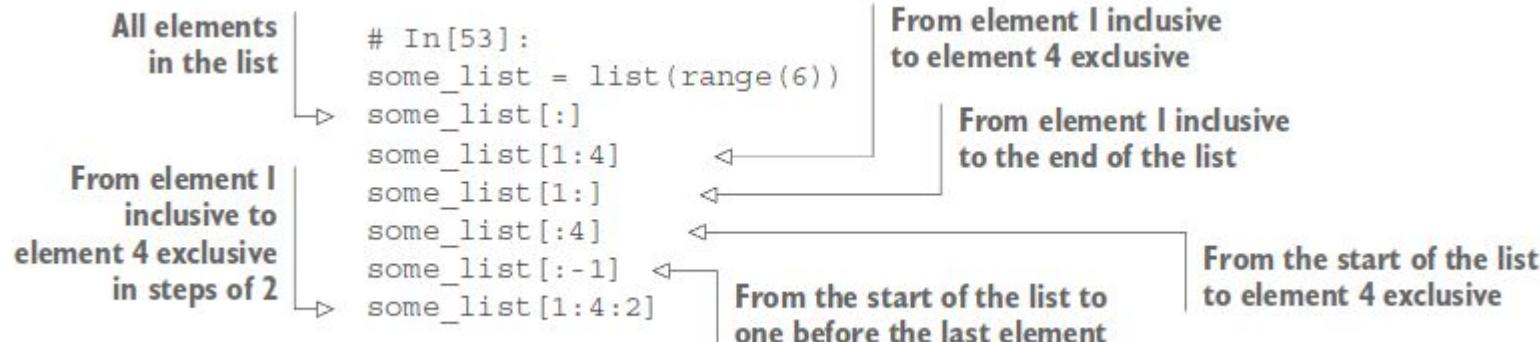
```
double_points = torch.ones(10, 2, dtype=torch.double)
int_points = torch.ones(10, 2, dtype=torch.int32)
short_points = torch.tensor([[1, 2], [3, 4]], dtype=torch.short)

double_points = torch.zeros(10, 2).double()
short_points = torch.ones(10, 2).short()

double_points = torch.zeros(10, 2).to(torch.double)
short_points = torch.ones(10, 2).to(dtype=torch.short)
int_points = int_points.to(torch.int16)
```

PyTorch :Tensor

Адресация



PyTorch :Tensor

Перенос на GPU

Возможность переноса на GPU является основной особенностью тензора PyTorch.

Это дает производить делать быстрые параллельные вычисления.

Существует несколько способов это сделать



PyTorch :Tensor

```
points_gpu = torch.tensor([[1.0, 4.0], [2.0, 1.0], [3.0, 4.0]], device='cuda')
```

Перенос на GPU

```
points_cpu = torch.tensor([[1.0, 4.0], [2.0, 1.0], [3.0, 4.0]])
points_cpu = torch.tensor([[1.0, 4.0], [2.0, 1.0], [3.0, 4.0]], device='cpu')

to_gpu = points_cpu.to(device='cuda')
to_gpu = points_cpu.to(device='cuda:0')
to_cpu = to_gpu.to(device='cpu')

points = torch.tensor([1, 2], device='cpu')
points_gpu = points.cuda()
points_gpu = points.cuda(0)
points_cpu = points_gpu.cpu()
```

PyTorch :Tensor

Операции условно
можно разделить на:

- **Creation** - создание тензора
- **Indexing, slicing, joining, transforming** - индексация, слайсинг, трансформации
- **Math** - математические операции - как правило **pointwise**, такие как log, exp etc ..
- **Reduction** - операции агрегации - mean, std, norm etc ..
- **Comparison** - min max etc ..
- **Random sampling**
- **Serialization**

PyTorch :Tensor

Операции

Операции доступны как непосредственно из пакета torch так и применимы непосредственно к тензору

Более подробно смотрите:

<http://pytorch.org/docs>

PyTorch :Tensor

Slicing

Операции

Slicing

```
[ ] #Индексация
    t3 = torch.tensor([[1, 2, 3],[3, 3, 6],[6, 7, 8]])
    print('t3 : ',t3)
    t3[:2,:2] #slicing
    print(t3[:2,:2])

⇒ t3 : tensor([[1, 2, 3],
               [3, 3, 6],
               [6, 7, 8]])
    tensor([[1, 2],
           [3, 3]])
```

PyTorch :Tensor

Операции

Joining

Joining

```
[ ] t4 = torch.tensor([[1, 1],[2, 2]])
# print ('t4.shape : ' , t4.shape)
print ('t4 : ' , t4)

t5 = torch.tensor([[3, 3],[4, 4]])
#print ('t5.shape : ' , t5.shape)
print ('t5 : ' , t5)

t_cat = torch.cat((t4, t5))
print ('t_cat.shape : ' , t_cat.shape)
print ('t_cat : ' , t_cat)

t_stack = torch.stack((t4, t5))
print ('t_stack.shape : ' , t_stack.shape)
print ('t_stack : ' , t_stack)
```

PyTorch :Tensor

Операции

Joining

```
t4 : tensor([[1, 1],  
            [2, 2]])  
t5 : tensor([[3, 3],  
            [4, 4]])  
t_cat.shape : torch.Size([4, 2])  
t_cat : tensor([[1, 1],  
                [2, 2],  
                [3, 3],  
                [4, 4]])  
t_stack.shape : torch.Size([2, 2, 2])  
t_stack : tensor([[[1, 1],  
                  [2, 2]],  
  
                  [[3, 3],  
                   [4, 4]]])
```

PyTorch :Tensor

Операции

Представления

torch.reshape более прокачанная функция.

Returns a tensor with the same data and number of elements as input, but with the specified shape.

When possible, the returned tensor will be a view of input. Otherwise, it will be a copy. Contiguous inputs and inputs with compatible strides can be reshaped without copying, but you should not depend on the copying vs. viewing behavior.

```
t = torch.tensor([[1, 1],[2, 2]])
# print ('t4.shape : ' , t4.shape)
print ('t4 : ' , t)

t_reshaped = t.reshape(-1)
print('t_reshaped.shape : ',t_reshaped.shape)

t_view = t.view(size=(1,4))
print('t_view.shape : ',t_view.shape)
print('t_view : ',t_view)

t4 :  tensor([[1, 1],
              [2, 2]])
t_reshaped.shape :  torch.Size([4])
t_view.shape :  torch.Size([1, 4])
t_view :  tensor([[1, 1, 2, 2]])
```

PyTorch :Tensor

Операции

Математические

```
t = torch.tensor([[1, 1],[2, 2]]).to(torch.float32)
print('t : ',t)
t_log = t.log()
print('t_log : ',t_log)
t_exp = t.exp()
print('t_exp : ',t_exp)
```

```
t :  tensor([[1., 1.],
             [2., 2.]])
t_log :  tensor([[0.0000, 0.0000],
                 [0.6931, 0.6931]])
t_exp :  tensor([[2.7183, 2.7183],
                 [7.3891, 7.3891]])
```

PyTorch :Tensor

Операции

Агрегации и сравнения

```
t = torch.tensor([[1, 2, 3],[3, 3, 6],[6, 7, 8]]).to(torch.float)
t_max = t.max()
print('t_max : ',float(t_max))
t_mean
t_min = t.min()
print('t_min : ',float(t_min))

t_std = t.std()
print('t_std : ',float(t_std))

t_mean = t.mean()
print('t_mean : ',float(t_mean))

t_norm = t.norm()
print('t_norm : ',float(t_norm))

t_max :  8.0
t_min :  1.0
t_std :  2.4494898319244385
t_mean :  4.333333492279053
t_norm :  14.73091983795166
```

Deep Learning

Нейронная сеть своими руками:

Как мы видим пока это не очень сильно отличается от numpy !!!

```
import torch

dtype = torch.FloatTensor

N, D_in, H, D_out = 64, 1000, 100, 10

x = torch.randn(N, D_in).type(dtype)
y = torch.randn(N, D_out).type(dtype)
w1 = torch.randn(D_in, H).type(dtype)
w2 = torch.randn(H, D_out).type(dtype)

learning_rate = 1e-6
```

```
for i in range(1000):

    # Прямой проход
    h = x.mm(w1)
    h_relu = h.clamp(min=0)
    y_pred = h_relu.mm(w2)

    loss = (y_pred - y).pow(2).sum()

    # Обратный проход
    grad_y_pred = 2.0 * (y_pred - y)
    grad_w2 = h_relu.t().mm(grad_y_pred)
    grad_h_relu = grad_y_pred.mm(w2.t())
    grad_h = grad_h_relu.clone()
    grad_h[h < 0] = 0
    grad_w1 = x.t().mm(grad_h)

    w1 -= learning_rate * grad_w1
    w2 -= learning_rate * grad_w2
```

PyTorch :Autograd

Предоставляет возможность автоматического дифференцирования.

autograd.Variable:

Является основным классом, инкапсулирующим операции необходимые для реализации концепции графа вычислений. Содержит в себе результаты выполнения прямого и обратного проходов по графу

Содержит ссылки на данные и градиент:

.data

.grad

Содержит метод **.backward()** для осуществления обратного прохода по графу вычислений.

PyTorch : Нейронная сеть своими руками Теперь используем Autograd!

```
from torch.autograd import Variable

N, D_in, H, D_out = 64, 1000, 100, 10

x = Variable(torch.randn(N, D_in), requires_grad = False)
y = Variable(torch.randn(N, D_out), requires_grad = False)

w1 = Variable(torch.randn(D_in, H), requires_grad = True)
w2 = Variable(torch.randn(H, D_out), requires_grad = True)
```

```
learning_rate = 1e-6
for i in range(1000):

    # Прямой проход
    y_pred = x.mm(w1).clamp(min=0).mm(w2)
    loss = (y_pred - y).pow(2).sum()

    if w1.grad is not None:
        w1.grad.data.zero_()
    if w2.grad is not None:
        w2.grad.data.zero_()

    # Обратный проход
    loss.backward()

    w1.data -= learning_rate * w1.grad.data
    w2.data -= learning_rate * w2.grad.data

    print(loss)
```

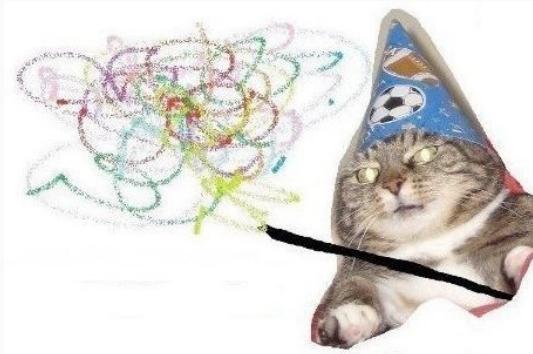
PyTorch : Нейронная сеть своими руками

Запустим теперь все на GPU!

Для этого нужно явно указать при инициализации переменных где вы хотите разместить ваши данные:

```
x = Variable(torch.randn(N, D_in).cuda(), requires_grad = False)
y = Variable(torch.randn(N, D_out).cuda(), requires_grad = False)

w1 = Variable(torch.randn(D_in, H).cuda(), requires_grad = True)
w2 = Variable(torch.randn(H, D_out).cuda(), requires_grad = True)
```



PyTorch : nn

**Высокоуровневая обертка для
построения нейронных сетей**

```
import torch
from torch.autograd import Variable

N, D_in, H, D_out = 64, 1000, 100, 10

x = Variable(torch.randn(N, D_in), requires_grad = False)
y = Variable(torch.randn(N, D_out), requires_grad = False)

model = torch.nn.Sequential(torch.nn.Linear(D_in, H),
                           torch.nn.ReLU(),
                           torch.nn.Linear(H, D_out))

loss_fn = torch.nn.MSELoss(size_average=False)
learning_rate = 1e-6

for i in range(10000):

    # Прямой проход
    y_pred = model(x)
    loss = loss_fn(y_pred, y)
    model.zero_grad()

    #Обратный проход
    loss.backward()

    for param in model.parameters():
        param.data -= learning_rate*param.grad.data
```

PyTorch : optim

**Реализует алгоритмы
оптимизации**

```
import torch
from torch.autograd import Variable

N, D_in, H, D_out = 64, 1000, 100, 10

x = Variable(torch.randn(N, D_in), requires_grad = False)
y = Variable(torch.randn(N, D_out), requires_grad = False)

model = torch.nn.Sequential(torch.nn.Linear(D_in, H),
                           torch.nn.ReLU(),
                           torch.nn.Linear(H, D_out))

loss_fn = torch.nn.MSELoss(size_average=False)

learning_rate = 1e-6
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

PyTorch : optim

**Реализует алгоритмы
оптимизации**

```
for i in range(10000):
    # Прямой проход
    y_pred = model(x)
    loss = loss_fn(y_pred, y)
    #Обнуляем веса градиентов
    optimizer.zero_grad()
    #Обратный проход
    loss.backward()
    #Обновляем параметры
    optimizer.step()
```

PyTorch :

**Давайте наконец
запустим все на GPU!**

```
import torch
from torch.autograd import Variable

N, D_in, H, D_out = 64, 1000, 100, 10

x = Variable(torch.randn(N, D_in).cuda(), requires_grad = False)
y = Variable(torch.randn(N, D_out).cuda(), requires_grad = False)

model = torch.nn.Sequential(torch.nn.Linear(D_in, H),
                           torch.nn.ReLU(),
                           torch.nn.Linear(H, D_out))

model.cuda()

loss_fn = torch.nn.MSELoss(size_average=False)

learning_rate = 1e-6
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

PyTorch :

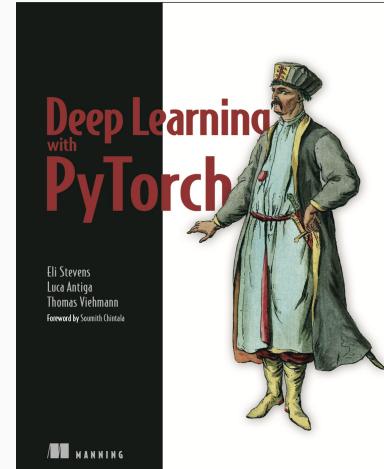
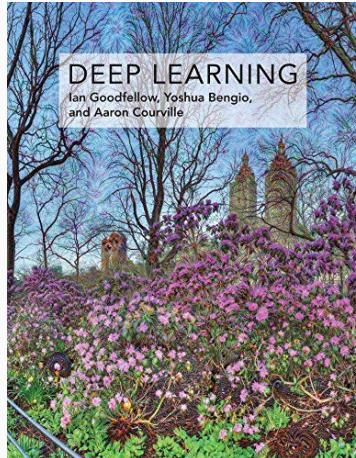
**Давайте наконец
запустим все на GPU!**

```
for i in range(10000):

    # Прямой проход
    y_pred = model(x)
    loss = loss_fn(y_pred, y)
    #Обнуляем веса градиентов
    optimizer.zero_grad()
    #Обратный проход
    loss.backward()
    #Обновляем параметры
    optimizer.step()
```

Deep Learning

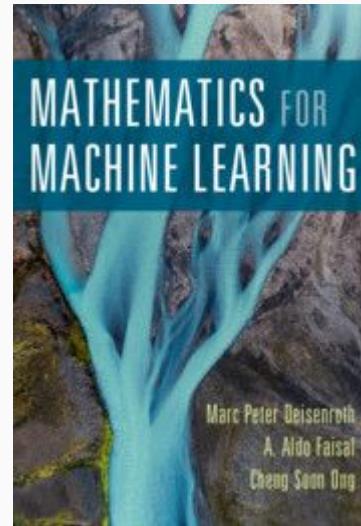
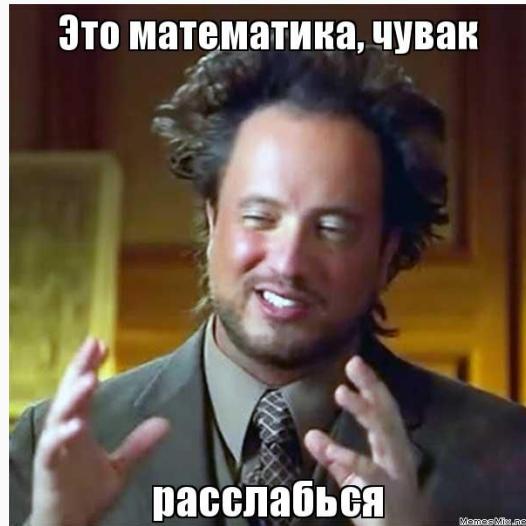
Книги (Deep Learning)



Deep Learning

Книги Математика (Если захотите копнуть глубже)

<https://mml-book.github.io/>



Почему Deep Learning

Курсы

<https://www.deeplearning.ai/>

<https://dlcourse.ai/>

<https://stepik.org/course/50352/promo>