

Лекция 6:

Инструментарий глубокого обучения

Железо, Динамические и статические графы, PyTorch и TensorFlow

Объявление:

Контрольная работа на 45 минут – скоро!

Три задачи:

1. Расчет функции потерь по матрице оценок классификатора, функция потерь или SoftMax или SVM.
2. Расчет прямого и обратного распространения по графу сети.
3. Расчет выхода для сверточной сети.

Данные по нескольким вариантам.

Задача на дом:

Входное изображение: CONV фильтр:

[1 2 3 4 5]
[2 2 1 1 1]
[3 2 1 1 1]
[4 1 1 1 1]
[5 1 1 1 1]

[0 -1 0]
[1 1 1]
[0 -1 0]

Посчитать выход сети: conv(depth=1, stride=2) -> ReLU -> MaxPool

Решение:

$$1) \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 2 & 1 & 1 & 1 \\ 3 & 2 & 1 & 1 & 1 \\ 4 & 1 & 1 & 1 & 1 \\ 5 & 1 & 1 & 1 & 1 \end{bmatrix} \otimes \begin{bmatrix} 0 & -1 & 0 \\ 1 & 1 & 1 \\ 0 & -1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & -2 \\ 3 & 1 \end{bmatrix}$$

↑
conv(stride2)

$$2) \text{ReLU} \left(\begin{bmatrix} 1 & -2 \\ 3 & 1 \end{bmatrix} \right) = \begin{bmatrix} 1 & 0 \\ 3 & 1 \end{bmatrix}$$

$$3) \text{MaxPool}(2,2) \left(\begin{bmatrix} 1 & 0 \\ 3 & 1 \end{bmatrix} \right) = \underline{\underline{3}}$$

Еще примеры задач:

Матрица оценок
классификатора:

[2.1 1.6 2.1]
[3.0 3.2 2.8]
[-2 3.7 3.8]

Посчитать:

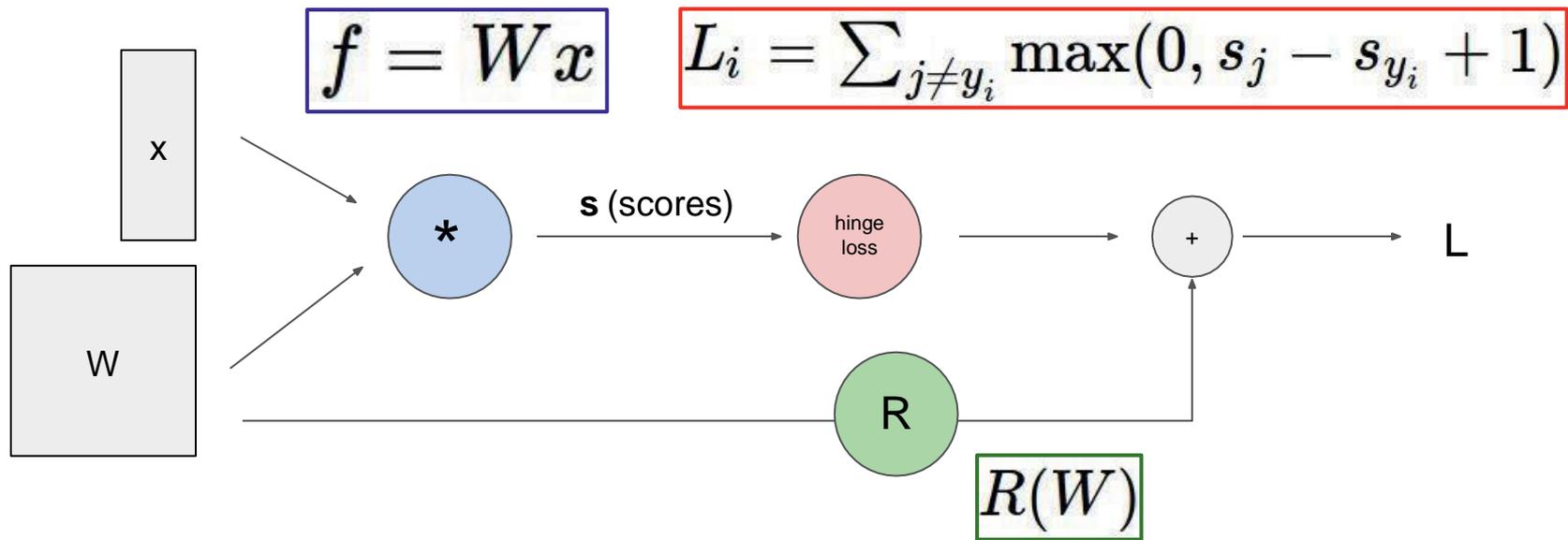
1. Функцию потерь мультиклассового SVM
2. Функцию потерь для SoftMax

3. Для заданной функции и входов посчитать прямое
и обратное распространение по сети.

При обратном распространении на входе считать градиент равным 1.

$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}} \quad \begin{array}{l} w_0 = 1, w_1 = -2, w_2 = 1 \\ x_0 = -1, x_1 = 1 \end{array}$$

Computational graphs



Вспоминаем...

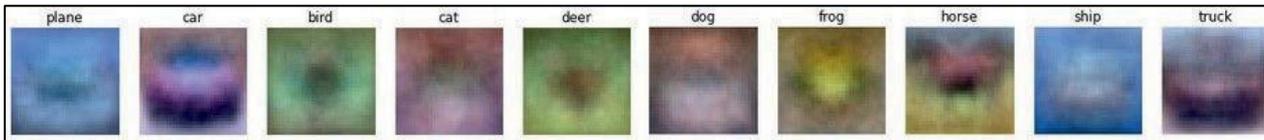
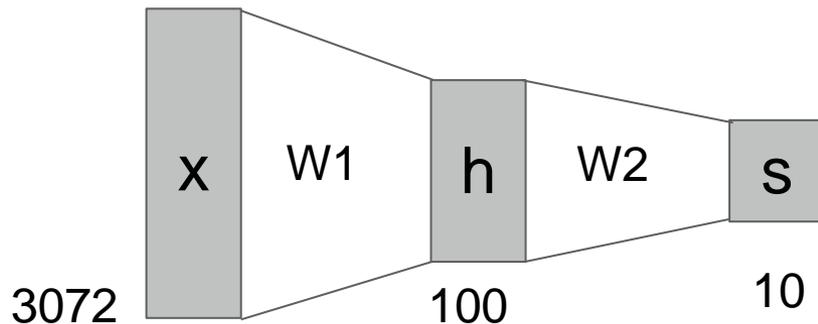
Neural Networks

Linear score function:

$$f = Wx$$

2-layer Neural Network

$$f = W_2 \max(0, W_1 x)$$



Вспоминаем...

Convolutional Neural Networks

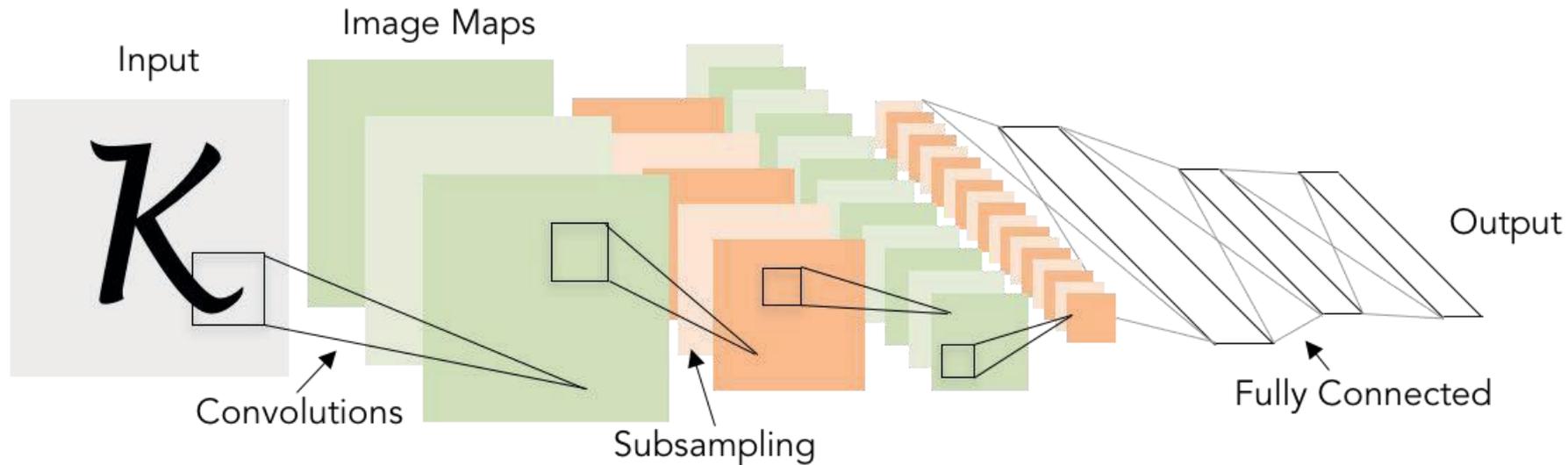
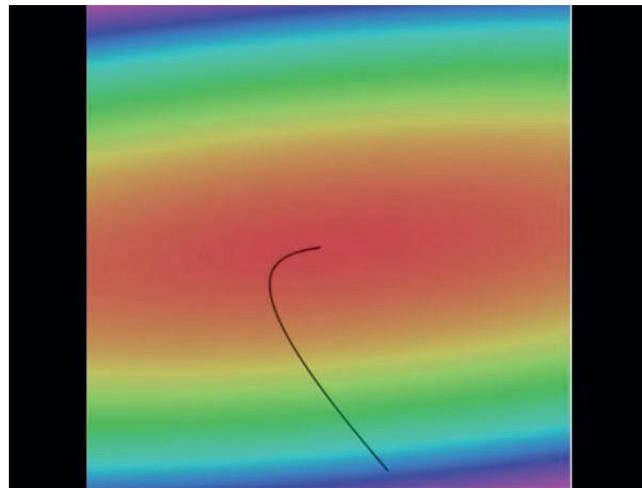
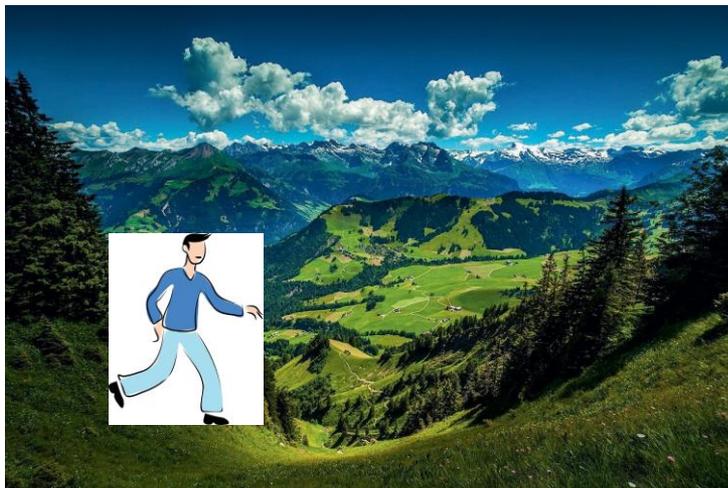


Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

Вспоминаем...

Learning network parameters through optimization



```
# Vanilla Gradient Descent

while True:
    weights_grad = evaluate_gradient(loss_fun, data, weights)
    weights += - step_size * weights_grad # perform parameter update
```

[Landscape image](#) is [CC0 1.0](#) public domain

[Walking man image](#) is [CC0 1.0](#) public domain

Сегодня:

- Аппаратное обеспечение
 - CPU, GPU
- Программное обеспечение
 - PyTorch и TensorFlow
 - Static and Dynamic computation graphs -
Статические и динамические вычислительные графы

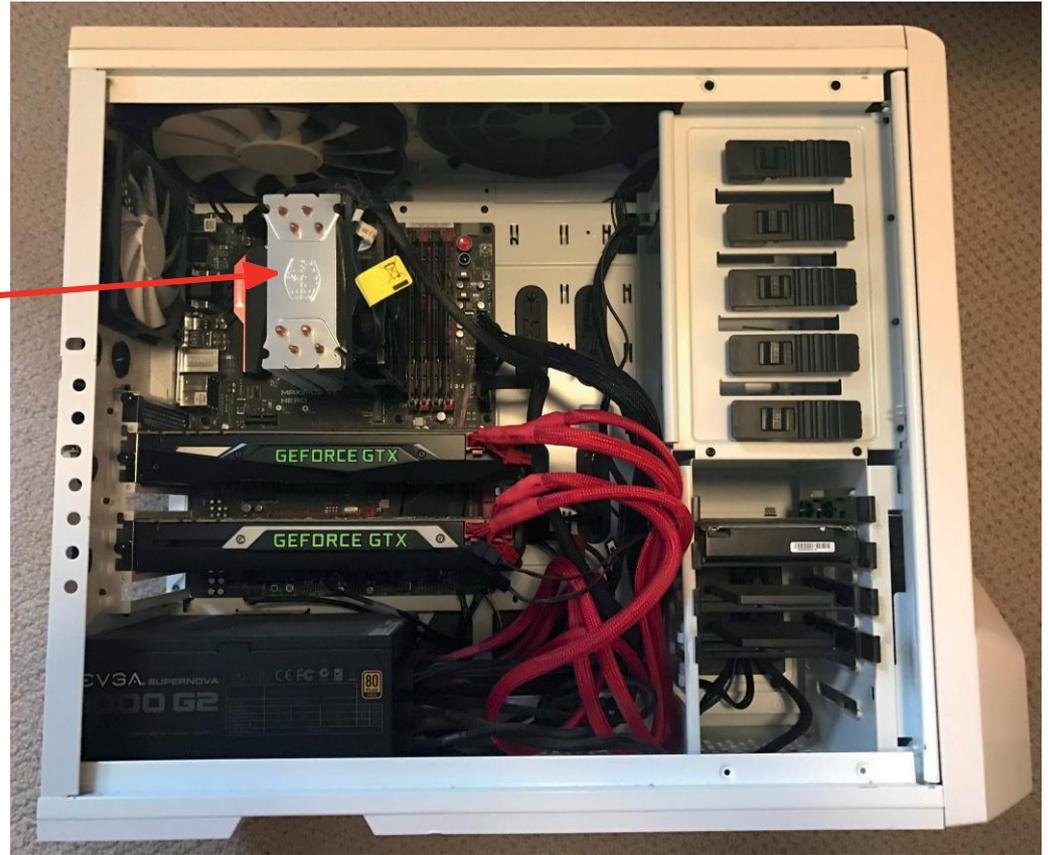
Deep Learning Hardware

CPU:

(central processing unit)



This image is licensed under [CC-BY 2.0](https://creativecommons.org/licenses/by/2.0/)



GPU:

(graphics processing unit)



[This image](#) is in the public domain

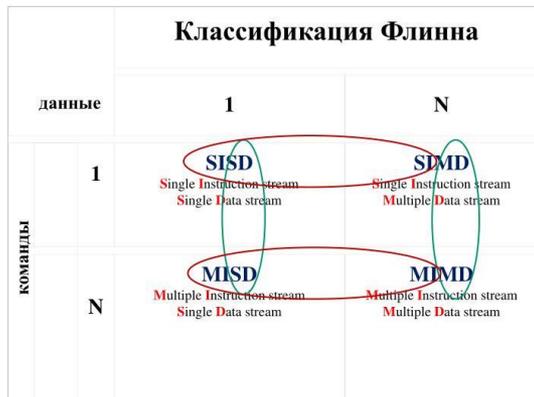


CPU vs GPU

	Cores	Clock Speed	Memory	Price	Speed
CPU (Intel Core i7-7700k)	4 (8 threads with hyperthreading)	4.2 GHz	System RAM	\$385	~540 GFLOPs FP32
GPU (NVIDIA RTX 2080 Ti)	3584	1.6 GHz	11 GB GDDR6	\$1199	~13.4 TFLOPs FP32

CPU: Поток до 100, но они универсальны

GPU: Поток до 10К, но они не универсальны
Векторная архитектура

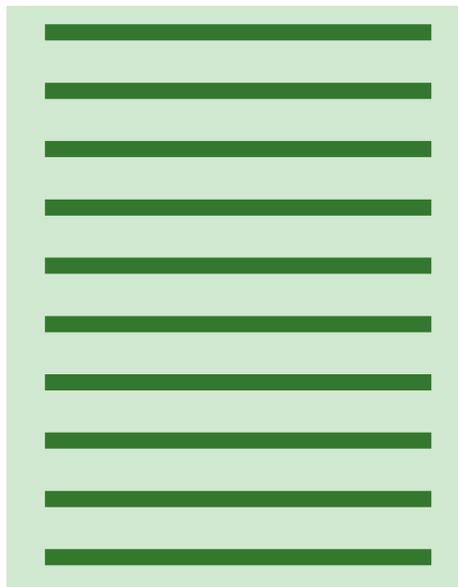


- GPU

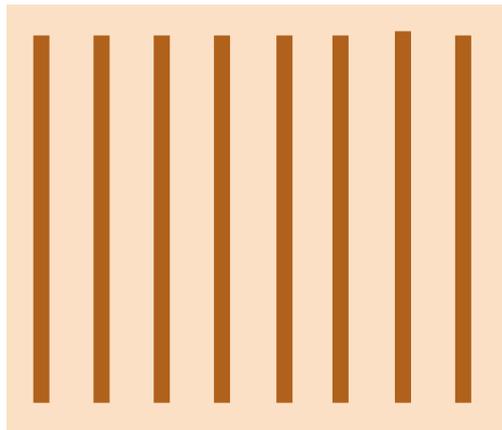
- CPU

Пример: умножение матриц

$A \times B$

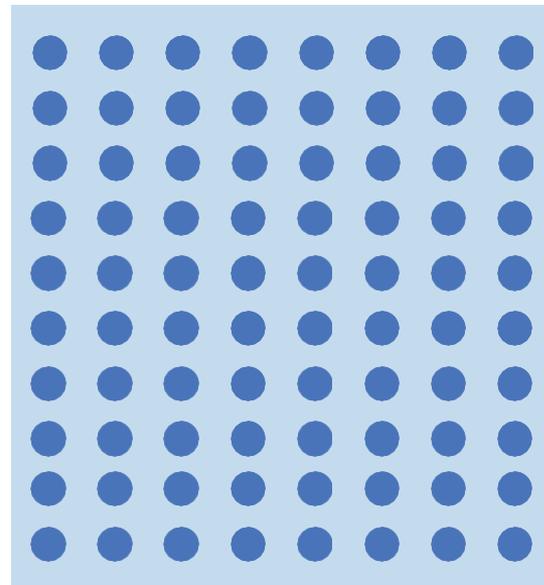


$B \times C$



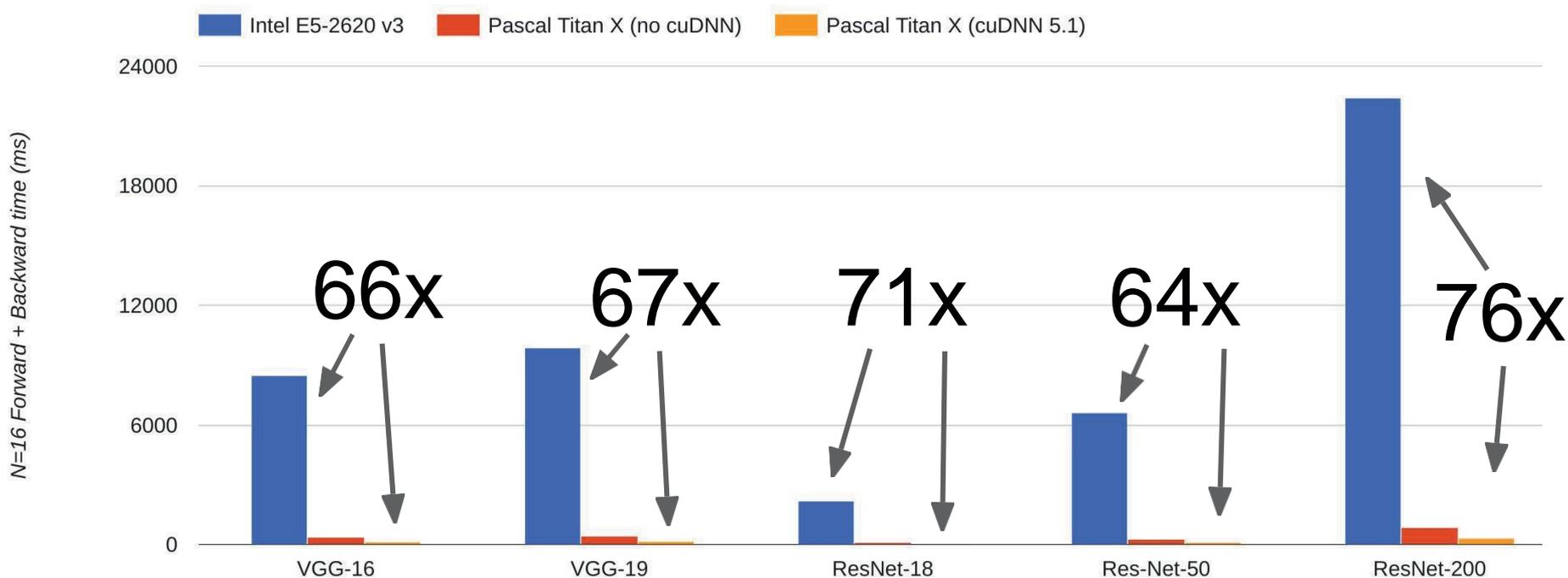
=

$A \times C$



CPU vs GPU

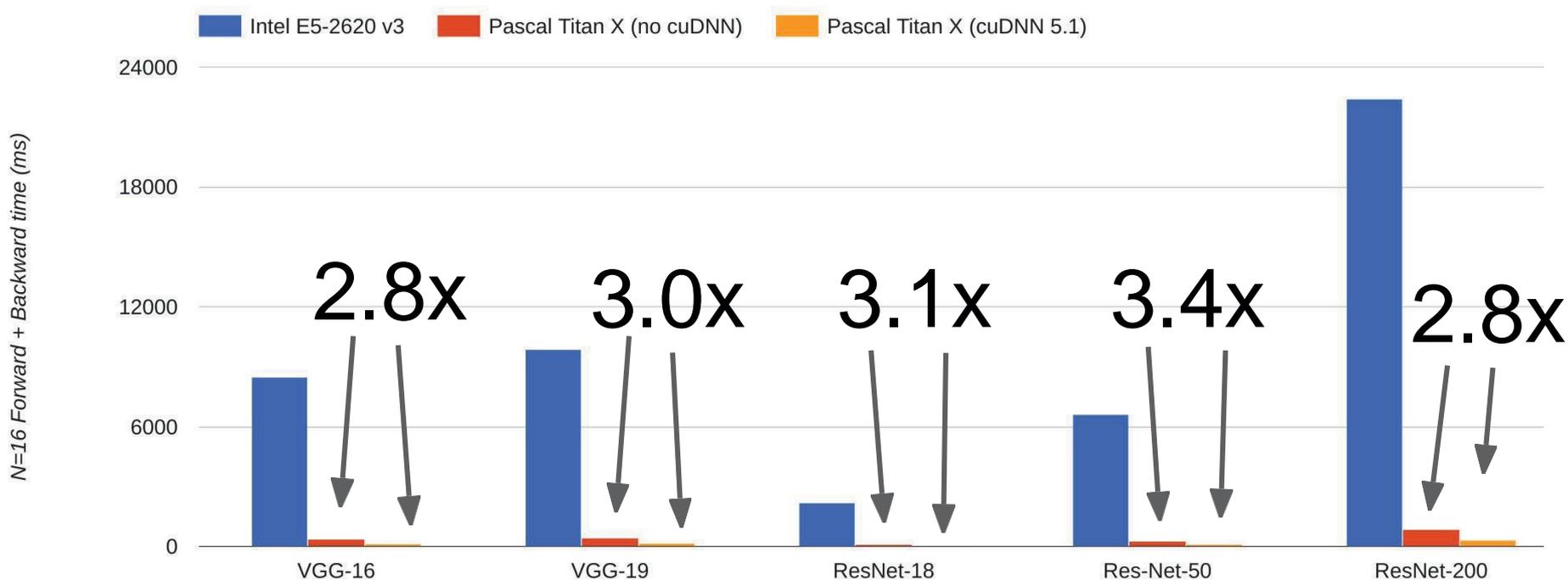
(CPU можно еще докрутить,
но это не важно)



Data from <https://github.com/jcjohnson/cnn-benchmarks>

CPU vs GPU:

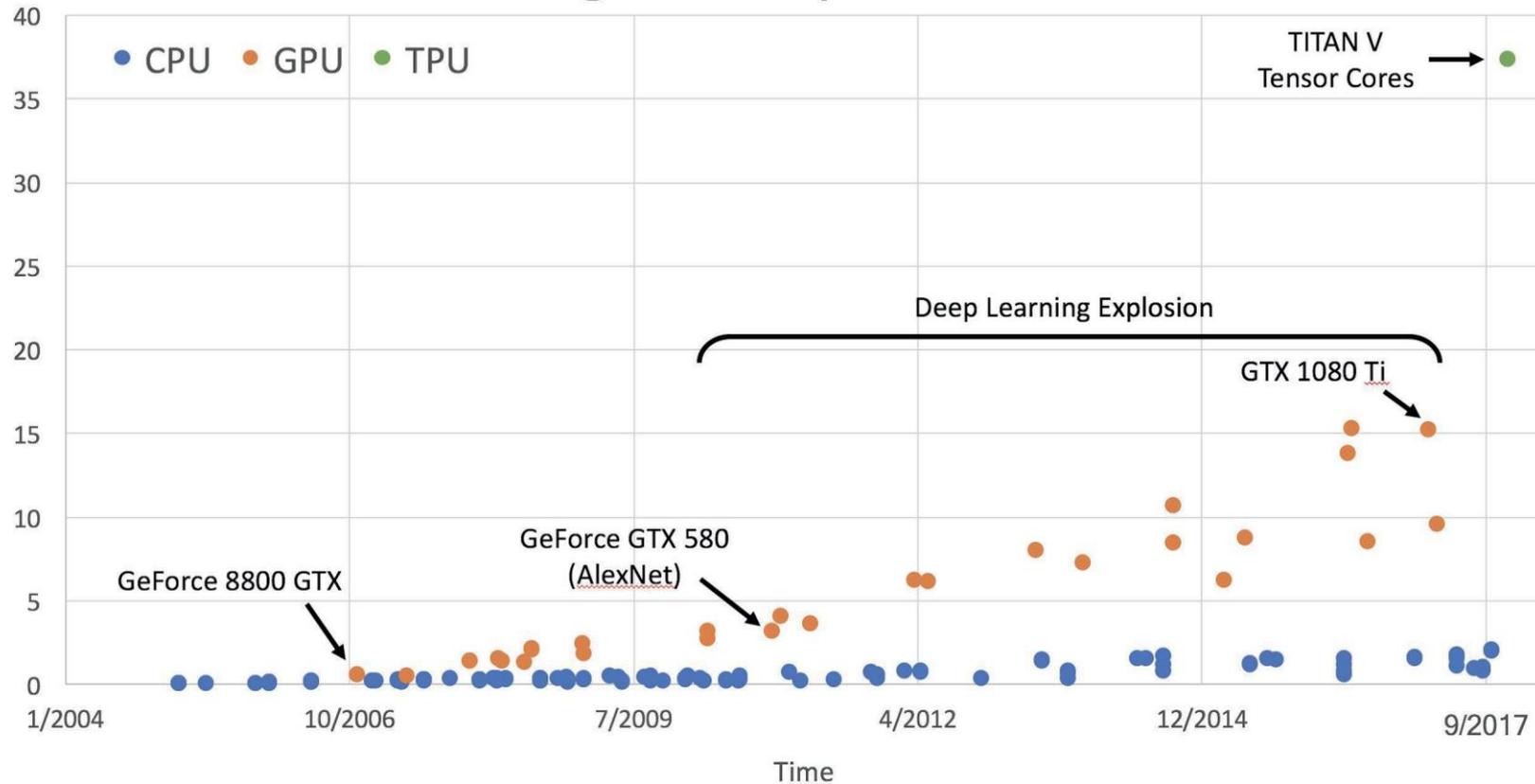
cuDNN сильно быстрее
чем просто CUDA



Data from <https://github.com/jcjohnson/cnn-benchmarks>

Более свежее - <https://www.aime.info/en/blog/deep-learning-gpu-benchmarks-2021/>

GigaFLOPs per Dollar



NVIDIA

vs

AMD

CPU vs GPU vs TPU

	Cores	Clock Speed	Memory	Price	Speed
CPU (Intel Core i7-7700k)	4 (8 threads with hyperthreading)	4.2 GHz	System RAM	\$385	~540 GFLOPs FP32
GPU NVIDIA RTX 2080 Ti	3584	1.6 GHz	11 GB GDDR6	\$1099	~13 TFLOPs FP32 ~114 TFLOPs FP16
GPU (Data Center) NVIDIA V100	5120 CUDA, 640 Tensor	1.5 GHz	16/32 GB HBM2	\$2.5/hr (GCP)	~8 TFLOPs FP64 ~16 TFLOPs FP32 ~125 TFLOPs FP16
TPU Google Cloud TPUv3	2 Matrix Units (MXUs) per core, 4 cores	?	128 GB HBM	\$8/hr (GCP)	~420 TFLOPs (non-standard FP)

CPU: Поток до 100, но они универсальны

GPU: Поток до 10К, но они не универсальны

TPU – сделаны под глубокое обучение:
Примеры – Intel, M2, NPU на борту телефонов

Программирование ГПУ

- CUDA (NVIDIA)
 - С-подобный код исполняемый на GPU
 - Рантайм обвес – драйвера, CUDA SDK
 - Оптимизация рантайм APIs: cuBLAS, cuFFT, cuDNN, etc
- OpenCL, Intel OpenVino
 - Похоже на CUDA, но можно запускать веде
 - Медленнее на NVIDIA
 - Плохо поддерживается в Torch/TF
- HIP <https://github.com/ROCm-Developer-Tools/HIP>
 - Пример кросскомпилятора
- Курс по CUDA CS 149: <http://cs149.stanford.edu/fall19/>

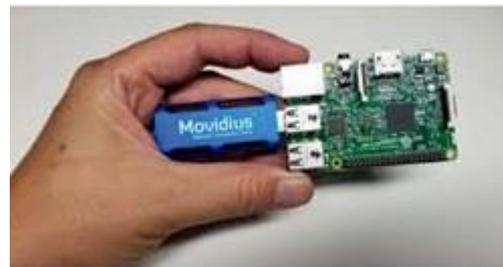
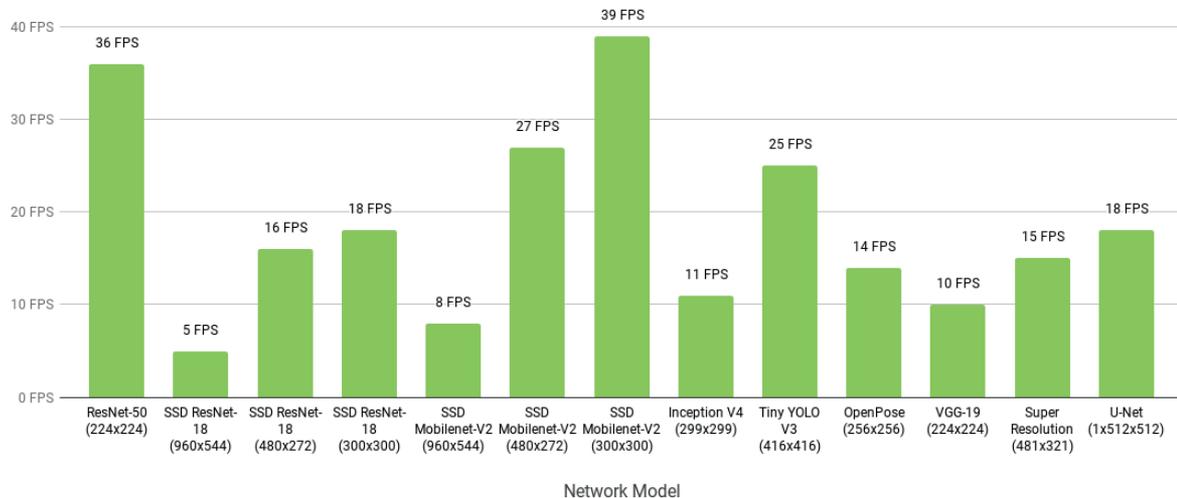
Inference Hardware

Железо для прода

High performance inference

Deep Learning Inference Performance

Jetson Nano (FP16, batch size 1)



Железо для прода

НПЦ Модуль Neuromatrix

Таблица 1.1 - FPS

	MC121.01	NMStick	MC127.05 NMCARD	и MC127.05 NMCARD batch- mode*
alexnet (227x227)	3,45	3,2	12,6	13
inception (299x299)	v3 0,63	0,6	8,12	12,43
inception (512x512)	v3 0,24	0,23	3,93	5,44
resnet (224x224)	18 2,28	2,2	25	47
squeezenet (224x224)	8,3	8	74,4	100
yolo v2 (416x416)	tiny 1,16	1,1	21	30,4
yolo (416x416)	v3 0,1	0,09	3,7	4
yolo v3 (416x416)	tiny 1,44	1,38	25,3	33,3



github.com/RC-MODULE/nmpp

Авиабилеты Яandex Scopus preview - S... Deep Fake Science... MDPI | Peer Review IPSI-Huawei TechRe... Нихоноров Артем...

global.mnk template ++ 7 months ago

README.md

NMPP

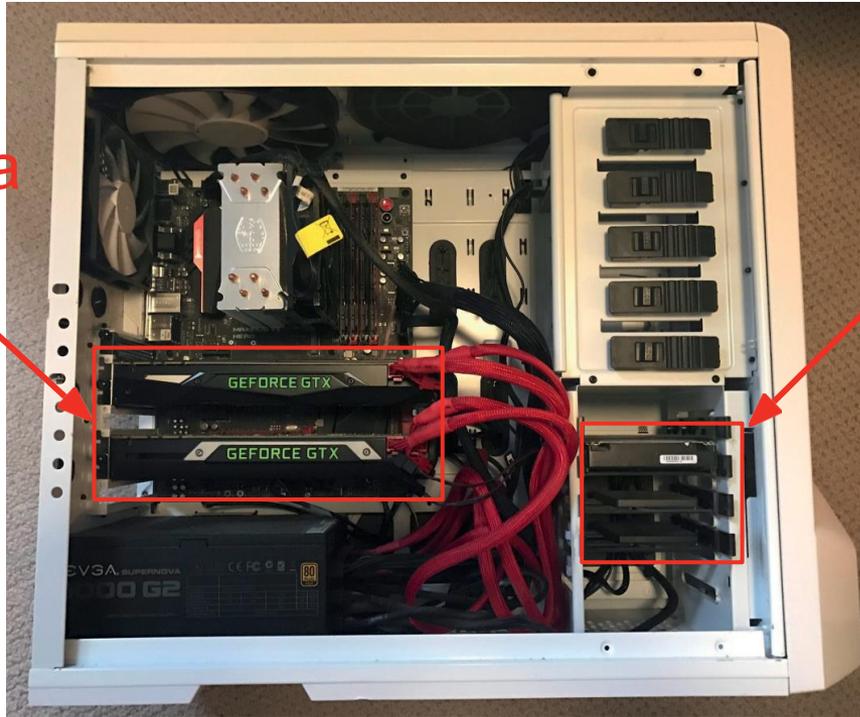
Документация:

HTML: <http://rc-module.github.io/nmpp/modules.html>
CHM(ZIP): <http://rc-module.github.io/nmpp/nmpp.zip>
CHM: <http://rc-module.github.io/nmpp/nmpp.chm> (При открытии необходимо снять галочку "Всегда спрашивать при открытии этого файла")
PDF: <http://rc-module.github.io/nmpp/nmpp.pdf>

Deep Learning Software

Коммуникация CPU и GPU

Моделька
здесь



А данные
здесь!

И система хранения крайне важна! – может стать узким местом

Решения:

- Загрузить все в ОЗУ
- SSD вместо HDD
- Загружать данные в МНОГО ПОТОКОВ

Зоопарк фреймворков!

Caffe
(UC Berkeley)



Caffe2
(Facebook)
mostly features absorbed
by PyTorch

Torch
(NYU / Facebook)



PyTorch
(Facebook)

Theano
(U Montreal)



TensorFlow
(Google)

Эти два наш фокус

PaddlePaddle
(Baidu)

Chainer
(Preferred Networks)
The company has officially migrated its research infrastructure to PyTorch

MXNet
(Amazon)

Developed by U Washington, CMU, MIT, Hong Kong U, etc but main framework of choice at AWS

CNTK
(Microsoft)

JAX
(Google)

And others...

Немного истории

Caffe - 2013, C++, декларативное описание сети, ModelZoo!
Tensorflow - 2015, питон, процедурное описание графа

Фрагмент AlexNet в формате Caffe:

```
1 name: "AlexNet"
2 layer {
3   name: "data"
4   type: "Data"
5   top: "data"
6   top: "label"
7   include {
8     phase: TRAIN
9   }
10  transform_param {
11    mirror: true
12    crop_size: 227
13    mean_file: "data/ilsrvrc12/imagenet_mean.binaryproto"
14  }
15  data_param {
16    source: "examples/imagenet/ilsrvrc12_train_lmdb"
17    batch_size: 256
18    backend: LHDB
19  }
20 }
21 layer {
22   name: "data"
23   type: "Data"
24   top: "data"
25   top: "label"
26   include {
27     phase: TEST
28   }
29  transform_param {
30    mirror: false
31    crop_size: 227
32    mean_file: "data/ilsrvrc12/imagenet_mean.binaryproto"
33  }
34  data_param {
35    source: "examples/imagenet/ilsrvrc12_val_lmdb"
36    batch_size: 50
37    backend: LHDB
38  }
39 }
40 layer {
41   name: "conv1"
42   type: "Convolution"
43   bottom: "data"
44   top: "conv1"
45   param {
46     lr_mult: 1
47     decay_mult: 1
48   }
49   param {
50     lr_mult: 2
51     decay_mult: 0
```

Пример Caffe ModelZoo:

Model Zoo

Sebastian Lapuschkin edited this page on 25 Apr 2019 · 122 revisions

Check out the [model zoo documentation](#) for details.

To acquire a model:

1. download the model gist by `./scripts/download_model_from_gist.sh <gist_id> <dirname>` to load the model metadata, architecture, solver configuration, and so on. (`<dirname>` is optional and defaults to `caffe/models`).
2. download the model weights by `./scripts/download_model_binary.py <model_dir>` where `<model_dir>` is the gist directory from the first step.

or visit the [\[model zoo documentation\]](#) (http://caffe.berkeleyvision.org/model_zoo.html) for complete instructions.

Table of Contents

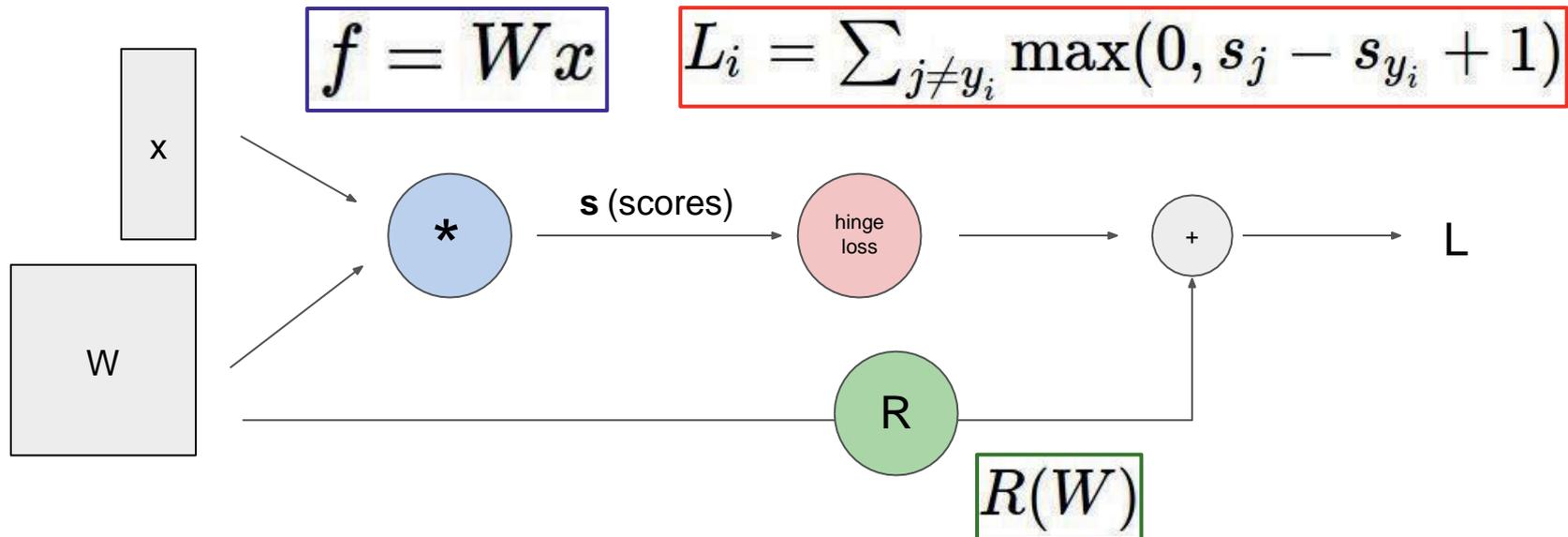
- [Berkeley-trained models](#)
- [Network in Network model](#)
- [Models from the BMVC-2014 paper "Return of the Devil in the Details: Delving Deep into Convolutional Nets"](#)

Models from the BMVC-2014 paper "Return of the Devil in the Details: Delving Deep into Convolutional Nets"

The models are trained on the ILSVRC-2012 dataset. The details can be found on the [project page](#) or in the following [BMVC-2014 paper](#):

Return of the Devil in the Details: Delving Deep into Convolutional Nets
K. Chatfield, K. Simonyan, A. Vedaldi, A. Zisserman
British Machine Vision Conference, 2014 (arXiv ref. cs1405.3531)

Вспоминаем: Computational Graphs



Вспоминаем: Computational Graphs

input image

weights

loss

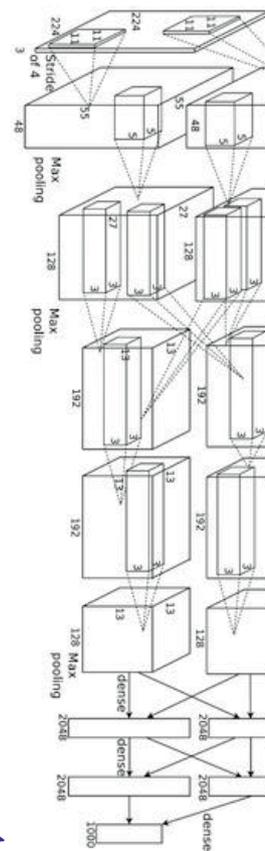


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Вспоминаем: Computational Graphs

input image

loss

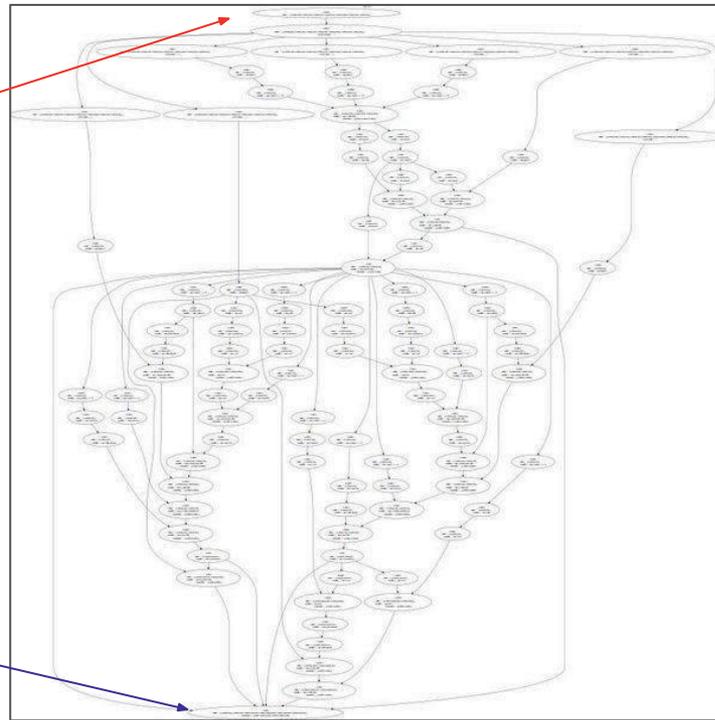


Figure reproduced with permission from a [Twitter post](#) by Andrej Karpathy.

Чего мы ждем от DL фреймворков?

- (1) Быстрая проверка новых идей
- (2) Автоматический расчет градиентов
- (3) Эффективное обучение и инференс на GPU
(с магией cuDNN, cuBLAS, OpenCL, etc)

Computational Graphs

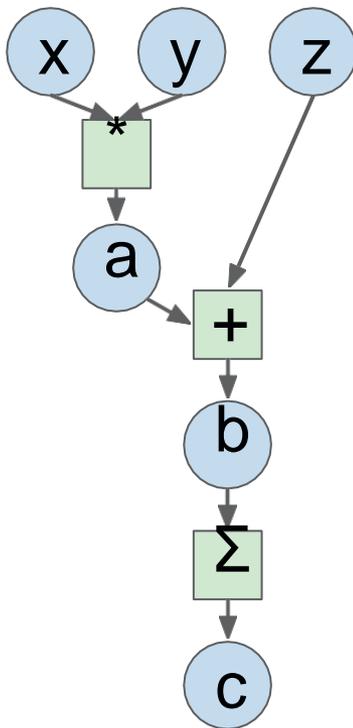
Numpy

```
import numpy as np
np.random.seed(0)

N, D = 3, 4

x = np.random.randn(N, D)
y = np.random.randn(N, D)
z = np.random.randn(N, D)

a = x * y
b = a + z
c = np.sum(b)
```



Computational Graphs

Numpy

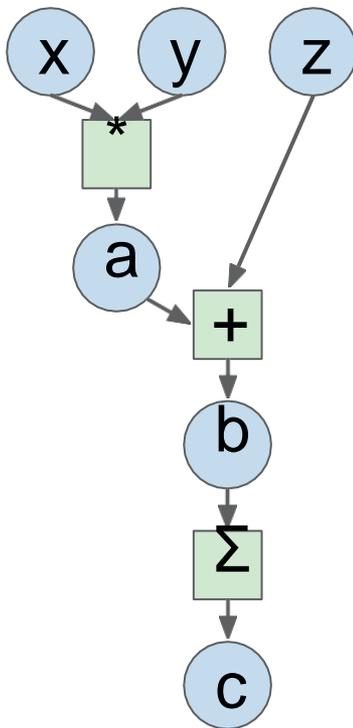
```
import numpy as np
np.random.seed(0)

N, D = 3, 4

x = np.random.randn(N, D)
y = np.random.randn(N, D)
z = np.random.randn(N, D)

a = x * y
b = a + z
c = np.sum(b)

grad_c = 1.0
grad_b = grad_c * np.ones((N, D))
grad_a = grad_b.copy()
grad_z = grad_b.copy()
grad_x = grad_a * y
grad_y = grad_a * x
```



Добавили расчет градиентов

Computational Graphs

Numpy

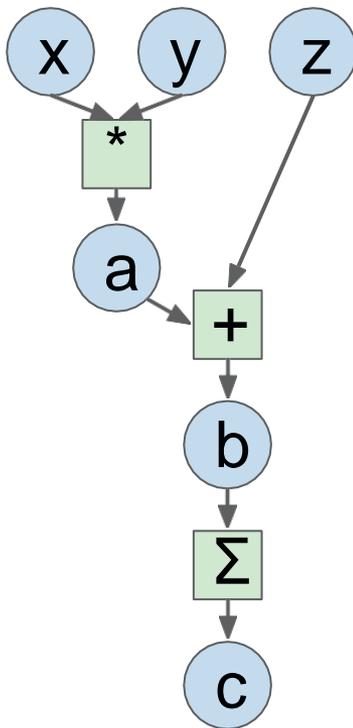
```
import numpy as np
np.random.seed(0)

N, D = 3, 4

x = np.random.randn(N, D)
y = np.random.randn(N, D)
z = np.random.randn(N, D)

a = x * y
b = a + z
c = np.sum(b)

grad_c = 1.0
grad_b = grad_c * np.ones((N, D))
grad_a = grad_b.copy()
grad_z = grad_b.copy()
grad_x = grad_a * y
grad_y = grad_a * x
```



Плюсы:

Просто и легко

Минусы:

- Градиенты надо писать самим
- Не заработает на GPU

Computational Graphs

Numpy

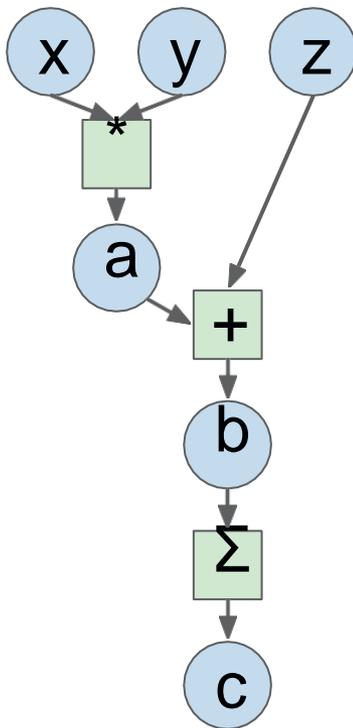
```
import numpy as np
np.random.seed(0)

N, D = 3, 4

x = np.random.randn(N, D)
y = np.random.randn(N, D)
z = np.random.randn(N, D)

a = x * y
b = a + z
c = np.sum(b)
```

```
grad_c = 1.0
grad_b = grad_c * np.ones((N, D))
grad_a = grad_b.copy()
grad_z = grad_b.copy()
grad_x = grad_a * y
grad_y = grad_a * x
```



PyTorch

```
import torch

N, D = 3, 4
x = torch.randn(N, D)
y = torch.randn(N, D)
z = torch.randn(N, D)

a = x * y
b = a + z
c = torch.sum(b)
```

А выглядит как numpy!

Computational Graphs

Numpy

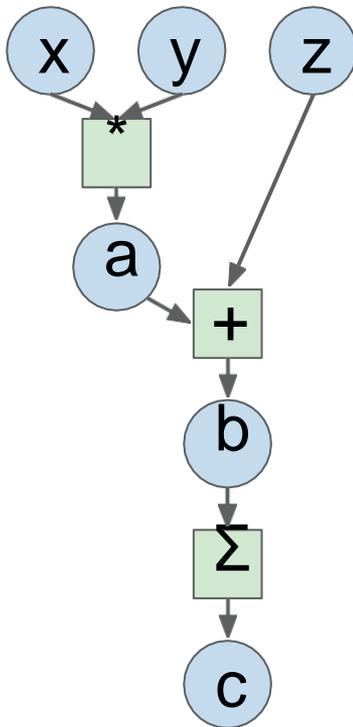
```
import numpy as np
np.random.seed(0)

N, D = 3, 4

x = np.random.randn(N, D)
y = np.random.randn(N, D)
z = np.random.randn(N, D)

a = x * y
b = a + z
c = np.sum(b)

grad_c = 1.0
grad_b = grad_c * np.ones((N, D))
grad_a = grad_b.copy()
grad_z = grad_b.copy()
grad_x = grad_a * y
grad_y = grad_a * x
```



PyTorch

```
import torch

N, D = 3, 4
x = torch.randn(N, D, requires_grad=True)
y = torch.randn(N, D)
z = torch.randn(N, D)

a = x * y
b = a + z
c = torch.sum(b)

c.backward()
print(x.grad)
```

PyTorch считает градиенты за нас!

Computational Graphs

Numpy

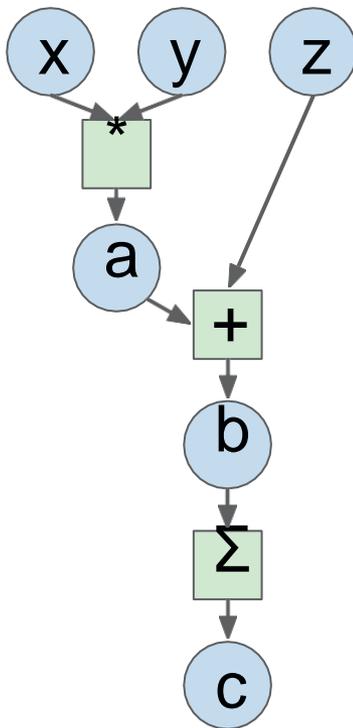
```
import numpy as np
np.random.seed(0)

N, D = 3, 4

x = np.random.randn(N, D)
y = np.random.randn(N, D)
z = np.random.randn(N, D)

a = x * y
b = a + z
c = np.sum(b)

grad_c = 1.0
grad_b = grad_c * np.ones((N, D))
grad_a = grad_b.copy()
grad_z = grad_b.copy()
grad_x = grad_a * y
grad_y = grad_a * x
```



PyTorch

```
import torch

device = 'cuda:0'
N, D = 3, 4
x = torch.randn(N, D, requires_grad=True,
                 device=device)
y = torch.randn(N, D, device=device)
z = torch.randn(N, D, device=device)

a = x * y
b = a + z
c = torch.sum(b)

c.backward()
print(x.grad)
```

И на GPU легко можно запустить!

PyTorch
(Подробнее)

PyTorch: Принципы

Tensor: Многомерный массив, как в numpy, но автоматически может обработаться и на GPU

Autograd: пакет автоматически формирующий графы обработки Тензоров и считает градиенты

Module: Слой нейронной сети, сохраняет состояние и веса

PyTorch: Версии

Здесь все про версию **PyTorch version 1.4**
(Released January 2020)

Все должно работать в версиях 1.x

<https://github.com/pytorch/pytorch>

PyTorch: Тензоры

Пример: Обучим
двухслойную сеть с ReLU
на случайных данных с L2
функцией потерь

```
import torch

device = torch.device('cpu')

N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in, device=device)
y = torch.randn(N, D_out, device=device)
w1 = torch.randn(D_in, H, device=device)
w2 = torch.randn(H, D_out, device=device)

learning_rate = 1e-6
for t in range(500):
    h = x.mm(w1)
    h_relu = h.clamp(min=0)
    y_pred = h_relu.mm(w2)
    loss = (y_pred - y).pow(2).sum()

    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: Тензоры

Создаем случайные
тензоры данных и весов



```
import torch

device = torch.device('cpu')

N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in, device=device)
y = torch.randn(N, D_out, device=device)
w1 = torch.randn(D_in, H, device=device)
w2 = torch.randn(H, D_out, device=device)

learning_rate = 1e-6
for t in range(500):
    h = x.mm(w1)
    h_relu = h.clamp(min=0)
    y_pred = h_relu.mm(w2)
    loss = (y_pred - y).pow(2).sum()

    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: Тензоры

Forward pass: расчет
предсказания
(prediction) и функции
потерь (loss)

```
import torch

device = torch.device('cpu')

N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in, device=device)
y = torch.randn(N, D_out, device=device)
w1 = torch.randn(D_in, H, device=device)
w2 = torch.randn(H, D_out, device=device)

learning_rate = 1e-6
for t in range(500):
    h = x.mm(w1)
    h_relu = h.clamp(min=0)
    y_pred = h_relu.mm(w2)
    loss = (y_pred - y).pow(2).sum()

    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: Тензоры

Backward pass: считаем
градиенты вручную



```
import torch

device = torch.device('cpu')

N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in, device=device)
y = torch.randn(N, D_out, device=device)
w1 = torch.randn(D_in, H, device=device)
w2 = torch.randn(H, D_out, device=device)

learning_rate = 1e-6
for t in range(500):
    h = x.mm(w1)
    h_relu = h.clamp(min=0)
    y_pred = h_relu.mm(w2)
    loss = (y_pred - y).pow(2).sum()

    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: Тензоры

```
import torch

device = torch.device('cpu')

N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in, device=device)
y = torch.randn(N, D_out, device=device)
w1 = torch.randn(D_in, H, device=device)
w2 = torch.randn(H, D_out, device=device)

learning_rate = 1e-6
for t in range(500):
    h = x.mm(w1)
    h_relu = h.clamp(min=0)
    y_pred = h_relu.mm(w2)
    loss = (y_pred - y).pow(2).sum()

    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
```

Шаг градиентного
спуска по весам



PyTorch: Тензоры

Чтобы повторить это на GPU просто скажите!

```
import torch
```

```
device = torch.device('cuda:0')
```

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

```
x = torch.randn(N, D_in, device=device)
```

```
y = torch.randn(N, D_out, device=device)
```

```
w1 = torch.randn(D_in, H, device=device)
```

```
w2 = torch.randn(H, D_out, device=device)
```

```
learning_rate = 1e-6
```

```
for t in range(500):
```

```
    h = x.mm(w1)
```

```
    h_relu = h.clamp(min=0)
```

```
    y_pred = h_relu.mm(w2)
```

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

```
    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

Создаем тензоры с флагом `requires_grad=True` чтобы включить autograd

Операции с тензорами при `requires_grad=True` заставляют PyTorch построить вычислительный граф

```
import torch

N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in)
y = torch.randn(N, D_out)
w1 = torch.randn(D_in, H, requires_grad=True)
w2 = torch.randn(H, D_out, requires_grad=True)

learning_rate = 1e-6
for t in range(500):
    y_pred = x.mm(w1).clamp(min=0).mm(w2)
    loss = (y_pred - y).pow(2).sum()

    loss.backward()

    with torch.no_grad():
        w1 -= learning_rate * w1.grad
        w2 -= learning_rate * w2.grad
        w1.grad.zero_()
        w2.grad.zero_()
```

PyTorch: Autograd

Для входа или выхода мы не будем считать градиенты

А для весов - будем

```
import torch

N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in)
y = torch.randn(N, D_out)
w1 = torch.randn(D_in, H, requires_grad=True)
w2 = torch.randn(H, D_out, requires_grad=True)

learning_rate = 1e-6
for t in range(500):
    y_pred = x.mm(w1).clamp(min=0).mm(w2)
    loss = (y_pred - y).pow(2).sum()

    loss.backward()

    with torch.no_grad():
        w1 -= learning_rate * w1.grad
        w2 -= learning_rate * w2.grad
        w1.grad.zero_()
        w2.grad.zero_()
```

PyTorch: Autograd

Прямой проход считаем как раньше, но без сохранения промежуточных значений - PyTorch сохраняет их в графе сам

```
import torch

N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in)
y = torch.randn(N, D_out)
w1 = torch.randn(D_in, H, requires_grad=True)
w2 = torch.randn(H, D_out, requires_grad=True)

learning_rate = 1e-6
for t in range(500):
    y_pred = x.mm(w1).clamp(min=0).mm(w2)
    loss = (y_pred - y).pow(2).sum()

    loss.backward()

    with torch.no_grad():
        w1 -= learning_rate * w1.grad
        w2 -= learning_rate * w2.grad
        w1.grad.zero_()
        w2.grad.zero_()
```

PyTorch: Autograd

Считаем градиент функции потерь по w_1 и w_2

```
import torch

N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in)
y = torch.randn(N, D_out)
w1 = torch.randn(D_in, H, requires_grad=True)
w2 = torch.randn(H, D_out, requires_grad=True)

learning_rate = 1e-6
for t in range(500):
    y_pred = x.mm(w1).clamp(min=0).mm(w2)
    loss = (y_pred - y).pow(2).sum()

    loss.backward()

    with torch.no_grad():
        w1 -= learning_rate * w1.grad
        w2 -= learning_rate * w2.grad
        w1.grad.zero_()
        w2.grad.zero_()
```

PyTorch: Autograd

Делаем шаг градиентного спуска, потом занулим градиенты. **Torch.no_grad** значит что эту часть в граф включать не нужно

```
import torch

N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in)
y = torch.randn(N, D_out)
w1 = torch.randn(D_in, H, requires_grad=True)
w2 = torch.randn(H, D_out, requires_grad=True)

learning_rate = 1e-6
for t in range(500):
    y_pred = x.mm(w1).clamp(min=0).mm(w2)
    loss = (y_pred - y).pow(2).sum()

    loss.backward()

    with torch.no_grad():
        w1 -= learning_rate * w1.grad
        w2 -= learning_rate * w2.grad
        w1.grad.zero_()
        w2.grad.zero_()
```

PyTorch: Autograd

Методы PyTorch с подчеркиванием меняют тензор in-place, не создавая нового тензора

```
import torch

N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in)
y = torch.randn(N, D_out)
w1 = torch.randn(D_in, H, requires_grad=True)
w2 = torch.randn(H, D_out, requires_grad=True)

learning_rate = 1e-6
for t in range(500):
    y_pred = x.mm(w1).clamp(min=0).mm(w2)
    loss = (y_pred - y).pow(2).sum()

    loss.backward()

    with torch.no_grad():
        w1 -= learning_rate * w1.grad
        w2 -= learning_rate * w2.grad
        w1.grad.zero_()
        w2.grad.zero_()
```

PyTorch: New Autograd Functions

Создадим свою функцию на основе autograd прописав методы forward() и backward() для тензоров

Объект ctx “кэширует” значения для backward прохода

Вспомогательная обертка для нового класса

```
class MyReLU(torch.autograd.Function):
    @staticmethod
    def forward(ctx, x):
        ctx.save_for_backward(x)
        return x.clamp(min=0)

    @staticmethod
    def backward(ctx, grad_y):
        x, = ctx.saved_tensors
        grad_input = grad_y.clone()
        grad_input[x < 0] = 0
        return grad_input

def my_relu(x):
    return MyReLU.apply(x)
```

PyTorch: New Autograd Functions

```
class MyReLU(torch.autograd.Function):  
    @staticmethod  
    def forward(ctx, x):  
        ctx.save_for_backward(x)  
        return x.clamp(min=0)  
  
    @staticmethod  
    def backward(ctx, grad_y):  
        x, = ctx.saved_tensors  
        grad_input = grad_y.clone()  
        grad_input[x < 0] = 0  
        return grad_input  
  
def my_relu(x):  
    return MyReLU.apply(x)
```

Можем ее использовать!
Но это нужно, **ТОЛЬКО** если
у вас действительно
необычный backward

```
N, D_in, H, D_out = 64, 1000, 100, 10  
  
x = torch.randn(N, D_in)  
y = torch.randn(N, D_out)  
w1 = torch.randn(D_in, H, requires_grad=True)  
w2 = torch.randn(H, D_out, requires_grad=True)  
  
learning_rate = 1e-6  
for t in range(500):  
    y_pred = my_relu(x.mm(w1)).mm(w2)  
    loss = (y_pred - y).pow(2).sum()  
  
    loss.backward()  
  
    with torch.no_grad():  
        w1 -= learning_rate * w1.grad  
        w2 -= learning_rate * w2.grad  
        w1.grad.zero_()  
        w2.grad.zero_()
```

PyTorch: nn

torch.nn обертка для
всего связанного с
СНС. Используйте его!

```
import torch

N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in)
y = torch.randn(N, D_out)

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

learning_rate = 1e-2
for t in range(500):
    y_pred = model(x)
    loss = torch.nn.functional.mse_loss(y_pred, y)

    loss.backward()

    with torch.no_grad():
        for param in model.parameters():
            param -= learning_rate * param.grad
    model.zero_grad()
```

PyTorch: nn

Определяйте модель как
последовательность обучаемых
слоев

```
import torch
```

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

```
x = torch.randn(N, D_in)
```

```
y = torch.randn(N, D_out)
```

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

```
learning_rate = 1e-2
```

```
for t in range(500):
```

```
    y_pred = model(x)
```

```
    loss = torch.nn.functional.mse_loss(y_pred, y)
```

```
    loss.backward()
```

```
    with torch.no_grad():
```

```
        for param in model.parameters():
```

```
            param -= learning_rate * param.grad
```

```
    model.zero_grad()
```

PyTorch: nn

Forward: подаем данные и считаем loss

```
import torch

N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in)
y = torch.randn(N, D_out)

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

learning_rate = 1e-2
for t in range(500):
    y_pred = model(x)
    loss = torch.nn.functional.mse_loss(y_pred, y)

    loss.backward()

    with torch.no_grad():
        for param in model.parameters():
            param -= learning_rate * param.grad
    model.zero_grad()
```

PyTorch: nn

Forward: подаем данные и считаем loss

torch.nn.functional содержит много полезных функций

```
import torch

N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in)
y = torch.randn(N, D_out)

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

learning_rate = 1e-2
for t in range(500):
    y_pred = model(x)
    loss = torch.nn.functional.mse_loss(y_pred, y)

    loss.backward()

    with torch.no_grad():
        for param in model.parameters():
            param -= learning_rate * param.grad
        model.zero_grad()
```

PyTorch: nn

Backward pass: считаем
градиент по всем весам (по
умолчанию все с флагом
`requires_grad=True`)

```
import torch

N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in)
y = torch.randn(N, D_out)

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

learning_rate = 1e-2
for t in range(500):
    y_pred = model(x)
    loss = torch.nn.functional.mse_loss(y_pred, y)
    loss.backward()

    with torch.no_grad():
        for param in model.parameters():
            param -= learning_rate * param.grad
    model.zero_grad()
```

PyTorch: nn

```
import torch

N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in)
y = torch.randn(N, D_out)

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

learning_rate = 1e-2
for t in range(500):
    y_pred = model(x)
    loss = torch.nn.functional.mse_loss(y_pred, y)

    loss.backward()

    with torch.no_grad():
        for param in model.parameters():
            param -= learning_rate * param.grad
        model.zero_grad()
```

Шаг градиентного спуска по весам (с запрещенным расчетом градиентов)

PyTorch: optim

optimizer из `torch.optim`
определяет все возможные
стратегии обучения

```
import torch

N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in)
y = torch.randn(N, D_out)

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

learning_rate = 1e-4
optimizer = torch.optim.Adam(model.parameters(),
                               lr=learning_rate)

for t in range(500):
    y_pred = model(x)
    loss = torch.nn.functional.mse_loss(y_pred, y)

    loss.backward()

    optimizer.step()
    optimizer.zero_grad()
```

PyTorch: optim

После расчета градиентов, optimizer делает шаг градиентного спуска и зануляет градиенты

```
import torch

N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in)
y = torch.randn(N, D_out)

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

learning_rate = 1e-4
optimizer = torch.optim.Adam(model.parameters(),
                               lr=learning_rate)

for t in range(500):
    y_pred = model(x)
    loss = torch.nn.functional.mse_loss(y_pred, y)

    loss.backward()

    optimizer.step()
    optimizer.zero_grad()
```

PyTorch: nn.Module

Новые слои

Torch.nn.Module класс реализующий слой НС с тензорами входе и выходе

Модули (слои) могут содержать веса или другие модули

Можно делать свои слои с autograd!

```
import torch

class TwoLayerNet(torch.nn.Module):
    def __init__(self, D_in, H, D_out):
        super(TwoLayerNet, self).__init__()
        self.linear1 = torch.nn.Linear(D_in, H)
        self.linear2 = torch.nn.Linear(H, D_out)

    def forward(self, x):
        h_relu = self.linear1(x).clamp(min=0)
        y_pred = self.linear2(h_relu)
        return y_pred

N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in)
y = torch.randn(N, D_out)

model = TwoLayerNet(D_in, H, D_out)

optimizer = torch.optim.SGD(model.parameters(), lr=1e-4)
for t in range(500):
    y_pred = model(x)
    loss = torch.nn.functional.mse_loss(y_pred, y)

    loss.backward()
    optimizer.step()
    optimizer.zero_grad()
```

PyTorch: nn.Module

НОВЫЕ СЛОИ

Определим свой
модуль на основе
Module

```
import torch
```

```
class TwoLayerNet(torch.nn.Module):  
    def __init__(self, D_in, H, D_out):  
        super(TwoLayerNet, self).__init__()  
        self.linear1 = torch.nn.Linear(D_in, H)  
        self.linear2 = torch.nn.Linear(H, D_out)  
  
    def forward(self, x):  
        h_relu = self.linear1(x).clamp(min=0)  
        y_pred = self.linear2(h_relu)  
        return y_pred
```



```
N, D_in, H, D_out = 64, 1000, 100, 10  
x = torch.randn(N, D_in)  
y = torch.randn(N, D_out)
```

```
model = TwoLayerNet(D_in, H, D_out)
```

```
optimizer = torch.optim.SGD(model.parameters(), lr=1e-4)  
for t in range(500):  
    y_pred = model(x)  
    loss = torch.nn.functional.mse_loss(y_pred, y)  
  
    loss.backward()  
    optimizer.step()  
    optimizer.zero_grad()
```

PyTorch: nn.Module

Новые слои

Создаем два слоя
(т.к. можно включать
другие модули)

```
import torch

class TwoLayerNet(torch.nn.Module):
    def __init__(self, D_in, H, D_out):
        super(TwoLayerNet, self).__init__()
        self.linear1 = torch.nn.Linear(D_in, H)
        self.linear2 = torch.nn.Linear(H, D_out)

    def forward(self, x):
        h_relu = self.linear1(x).clamp(min=0)
        y_pred = self.linear2(h_relu)
        return y_pred

N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in)
y = torch.randn(N, D_out)

model = TwoLayerNet(D_in, H, D_out)

optimizer = torch.optim.SGD(model.parameters(), lr=1e-4)
for t in range(500):
    y_pred = model(x)
    loss = torch.nn.functional.mse_loss(y_pred, y)

    loss.backward()
    optimizer.step()
    optimizer.zero_grad()
```

PyTorch: nn.Module

Новые слои

Прямой проход на
основе дочерних слоев

Обратный проход
автоматом, за счет
autograd

```
import torch

class TwoLayerNet(torch.nn.Module):
    def __init__(self, D_in, H, D_out):
        super(TwoLayerNet, self).__init__()
        self.linear1 = torch.nn.Linear(D_in, H)
        self.linear2 = torch.nn.Linear(H, D_out)
```

```
    def forward(self, x):
        h_relu = self.linear1(x).clamp(min=0)
        y_pred = self.linear2(h_relu)
        return y_pred
```

```
N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in)
y = torch.randn(N, D_out)
```

```
model = TwoLayerNet(D_in, H, D_out)
```

```
optimizer = torch.optim.SGD(model.parameters(), lr=1e-4)
for t in range(500):
    y_pred = model(x)
    loss = torch.nn.functional.mse_loss(y_pred, y)

    loss.backward()
    optimizer.step()
    optimizer.zero_grad()
```

PyTorch: nn.Module

Новые слои

Создаем и обучаем
экземпляр нашей
сетки



```
import torch

class TwoLayerNet(torch.nn.Module):
    def __init__(self, D_in, H, D_out):
        super(TwoLayerNet, self).__init__()
        self.linear1 = torch.nn.Linear(D_in, H)
        self.linear2 = torch.nn.Linear(H, D_out)

    def forward(self, x):
        h_relu = self.linear1(x).clamp(min=0)
        y_pred = self.linear2(h_relu)
        return y_pred
```

```
N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in)
y = torch.randn(N, D_out)
```

```
model = TwoLayerNet(D_in, H, D_out)

optimizer = torch.optim.SGD(model.parameters(), lr=1e-4)
for t in range(500):
    y_pred = model(x)
    loss = torch.nn.functional.mse_loss(y_pred, y)

    loss.backward()
    optimizer.step()
    optimizer.zero_grad()
```

PyTorch: nn.Module

Новые слои

Обычно смешивают использование собственных модулей на основе Module и последовательных (Sequential) контейнеров

```
import torch

class ParallelBlock(torch.nn.Module):
    def __init__(self, D_in, D_out):
        super(ParallelBlock, self).__init__()
        self.linear1 = torch.nn.Linear(D_in, D_out)
        self.linear2 = torch.nn.Linear(D_in, D_out)
    def forward(self, x):
        h1 = self.linear1(x)
        h2 = self.linear2(x)
        return (h1 * h2).clamp(min=0)

N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in)
y = torch.randn(N, D_out)

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

optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)
for t in range(500):
    y_pred = model(x)
    loss = torch.nn.functional.mse_loss(y_pred, y)
    loss.backward()
    optimizer.step()
    optimizer.zero_grad()
```

PyTorch: nn.Module

Новые слои

Определим свой
компонент как subclass
Module

```
import torch
```

```
class ParallelBlock(torch.nn.Module):  
    def __init__(self, D_in, D_out):  
        super(ParallelBlock, self).__init__()  
        self.linear1 = torch.nn.Linear(D_in, D_out)  
        self.linear2 = torch.nn.Linear(D_in, D_out)  
    def forward(self, x):  
        h1 = self.linear1(x)  
        h2 = self.linear2(x)  
        return (h1 * h2).clamp(min=0)
```

```
N, D_in, H, D_out = 64, 1000, 100, 10  
x = torch.randn(N, D_in)  
y = torch.randn(N, D_out)
```

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

```
optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)  
for t in range(500):  
    y_pred = model(x)  
    loss = torch.nn.functional.mse_loss(y_pred, y)  
    loss.backward()  
    optimizer.step()  
    optimizer.zero_grad()
```

PyTorch: nn.Module

НОВЫЕ СЛОИ

Соберем из этого сеть на основе torch.nn.Sequential



```
import torch

class ParallelBlock(torch.nn.Module):
    def __init__(self, D_in, D_out):
        super(ParallelBlock, self).__init__()
        self.linear1 = torch.nn.Linear(D_in, D_out)
        self.linear2 = torch.nn.Linear(D_in, D_out)

    def forward(self, x):
        h1 = self.linear1(x)
        h2 = self.linear2(x)
        return (h1 * h2).clamp(min=0)
```

```
N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in)
y = torch.randn(N, D_out)
```

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

```
optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)
for t in range(500):
    y_pred = model(x)
    loss = torch.nn.functional.mse_loss(y_pred, y)
    loss.backward()
    optimizer.step()
    optimizer.zero_grad()
```

PyTorch: Pretrained Models

Предобученные модели

Для компьютерного зрения <https://github.com/pytorch/vision>

Дифференцируемые модели компьютерного зрения:

<https://github.com/kornia/kornia>

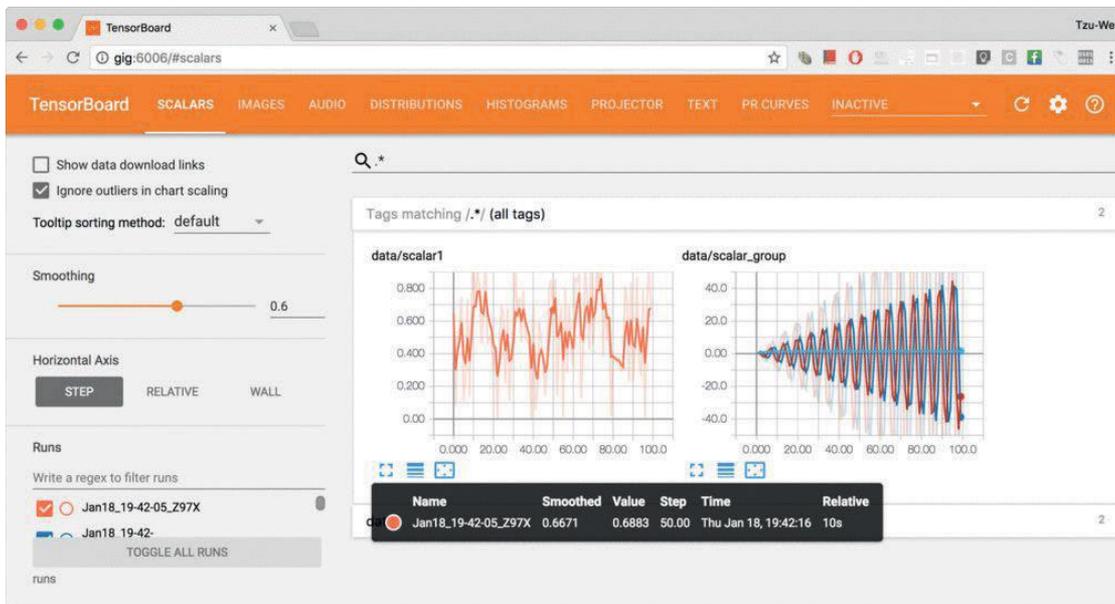
Пример: Канни - <https://kornia-tutorials.readthedocs.io/en/latest/canny.html>

```
import torch
import torchvision

alexnet = torchvision.models.alexnet(pretrained=True)
vgg16 = torchvision.models.vgg16(pretrained=True)
resnet101 = torchvision.models.resnet101(pretrained=True)
```

PyTorch: torch.utils.tensorboard

Визуализация процесса обучения.



[This image](#) is licensed under [CC-BY 4.0](#); no changes were made to the image

PyTorch: Computational Graphs

input image

loss

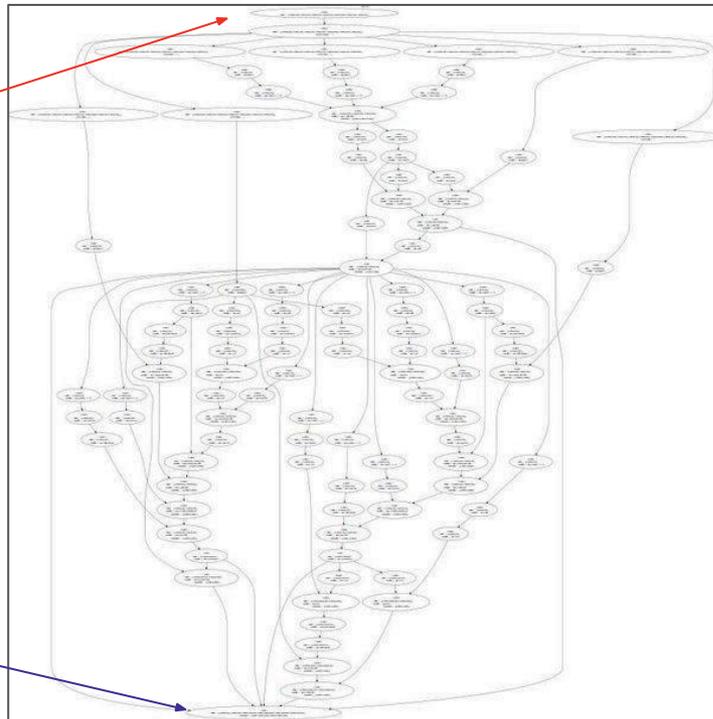


Figure reproduced with permission from a [Twitter post](#) by Andrej Karpathy.

PyTorch: Dynamic Computation Graphs

```
import torch

N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in)
y = torch.randn(N, D_out)
w1 = torch.randn(D_in, H, requires_grad=True)
w2 = torch.randn(H, D_out, requires_grad=True)

learning_rate = 1e-6
for t in range(500):
    y_pred = x.mm(w1).clamp(min=0).mm(w2)
    loss = (y_pred - y).pow(2).sum()

    loss.backward()
```

PyTorch: Dynamic Computation Graphs

x

w1

w2

y

```
import torch

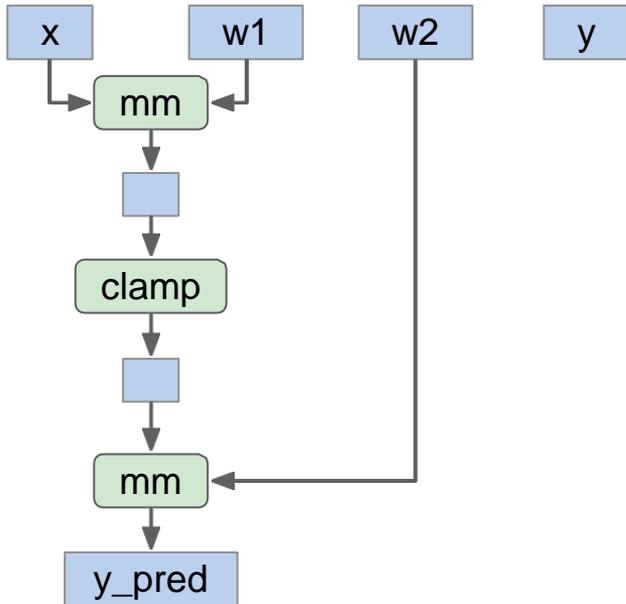
N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in)
y = torch.randn(N, D_out)
w1 = torch.randn(D_in, H, requires_grad=True)
w2 = torch.randn(H, D_out, requires_grad=True)

learning_rate = 1e-6
for t in range(500):
    y_pred = x.mm(w1).clamp(min=0).mm(w2)
    loss = (y_pred - y).pow(2).sum()

    loss.backward()
```

Создаем тензоры

PyTorch: Dynamic Computation Graphs



```
import torch

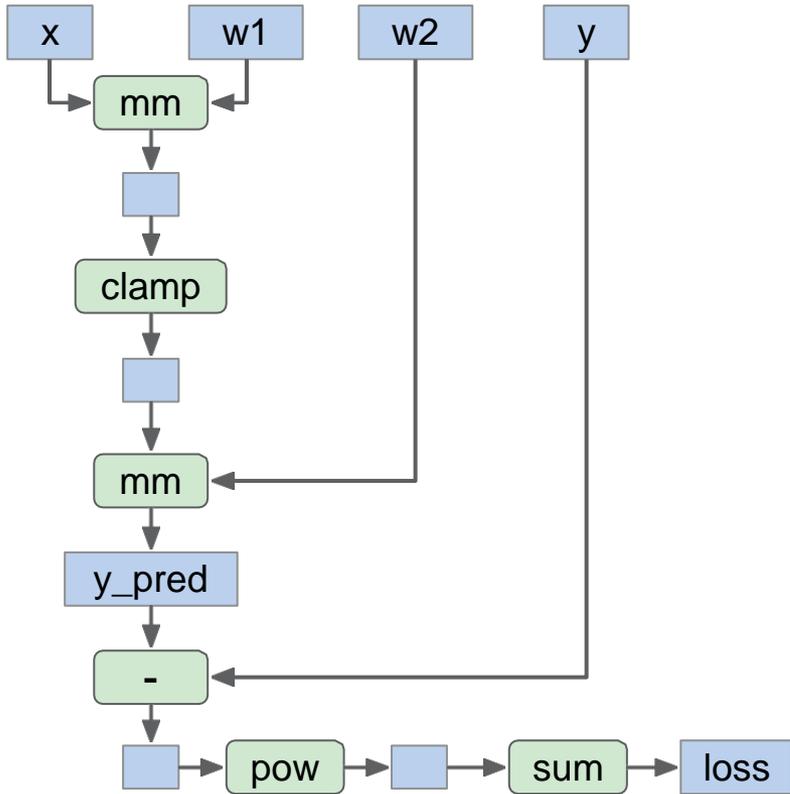
N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in)
y = torch.randn(N, D_out)
w1 = torch.randn(D_in, H, requires_grad=True)
w2 = torch.randn(H, D_out, requires_grad=True)

learning_rate = 1e-6
for t in range(500):
    y_pred = x.mm(w1).clamp(min=0).mm(w2)
    loss = (y_pred - y).pow(2).sum()

    loss.backward()
```

Зададим структуру графа И
выполним расчет

PyTorch: Dynamic Computation Graphs



```
import torch
```

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

```
x = torch.randn(N, D_in)
```

```
y = torch.randn(N, D_out)
```

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

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

```
learning_rate = 1e-6
```

```
for t in range(500):
```

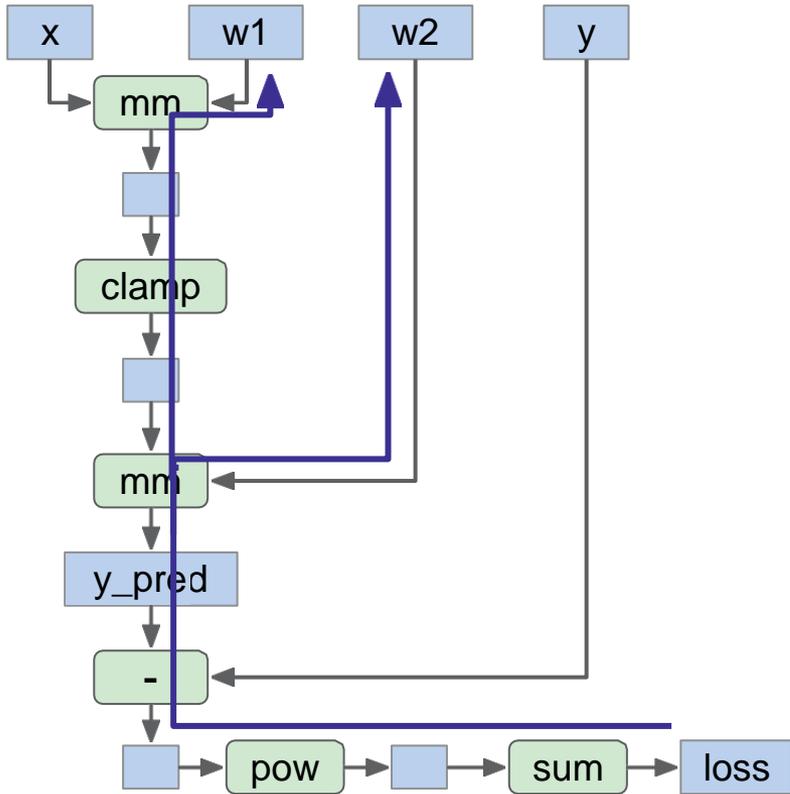
```
    y_pred = x.mm(w1).clamp(min=0).mm(w2)
```

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

```
    loss.backward()
```

Зададим структуру графа И
выполним расчет

PyTorch: Dynamic Computation Graphs



```
import torch

N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in)
y = torch.randn(N, D_out)
w1 = torch.randn(D_in, H, requires_grad=True)
w2 = torch.randn(H, D_out, requires_grad=True)

learning_rate = 1e-6
for t in range(500):
    y_pred = x.mm(w1).clamp(min=0).mm(w2)
    loss = (y_pred - y).pow(2).sum()

    loss.backward()
```

Выстраивается расчет между loss и w1, w2 и выполняются вычисления

PyTorch: Dynamic Computation Graphs

x

w1

w2

y

```
import torch

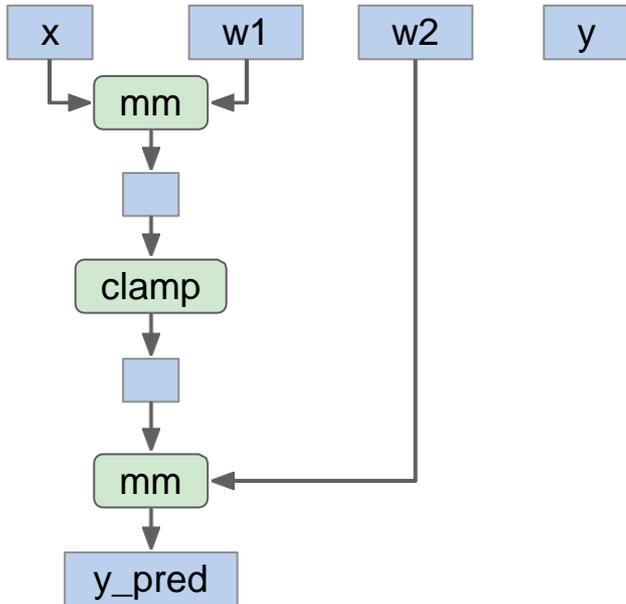
N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in)
y = torch.randn(N, D_out)
w1 = torch.randn(D_in, H, requires_grad=True)
w2 = torch.randn(H, D_out, requires_grad=True)

learning_rate = 1e-6
for t in range(500):
    y_pred = x.mm(w1).clamp(min=0).mm(w2)
    loss = (y_pred - y).pow(2).sum()

    loss.backward()
```

Для новой итерации цикла все снова!

PyTorch: Dynamic Computation Graphs



```
import torch

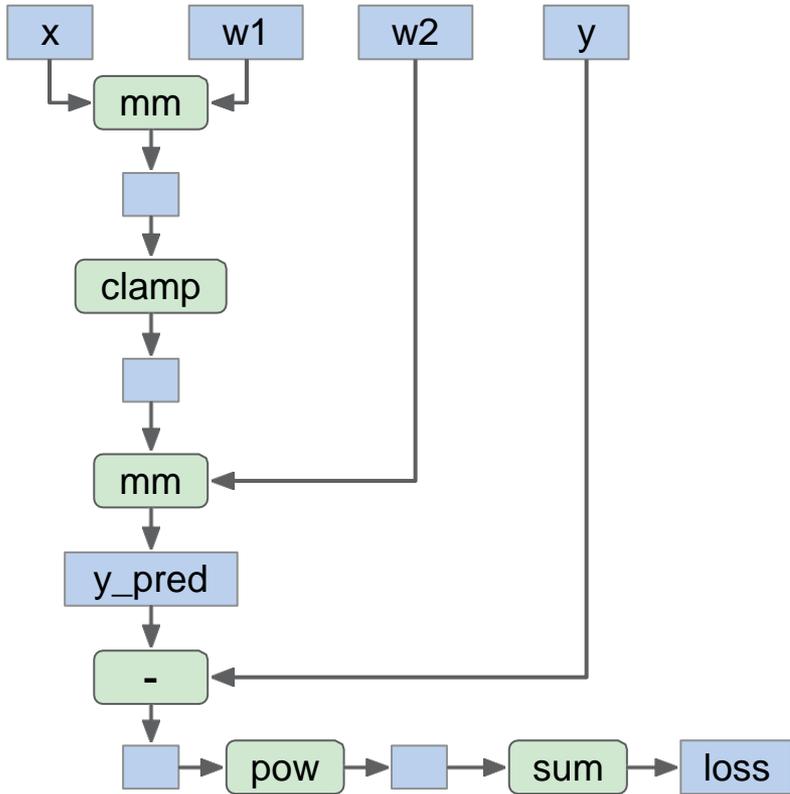
N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in)
y = torch.randn(N, D_out)
w1 = torch.randn(D_in, H, requires_grad=True)
w2 = torch.randn(H, D_out, requires_grad=True)

learning_rate = 1e-6
for t in range(500):
    y_pred = x.mm(w1).clamp(min=0).mm(w2)
    loss = (y_pred - y).pow(2).sum()

    loss.backward()
```

Зададим структуру графа И
выполним расчет

PyTorch: Dynamic Computation Graphs



```
import torch
```

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

```
x = torch.randn(N, D_in)
```

```
y = torch.randn(N, D_out)
```

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

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

```
learning_rate = 1e-6
```

```
for t in range(500):
```

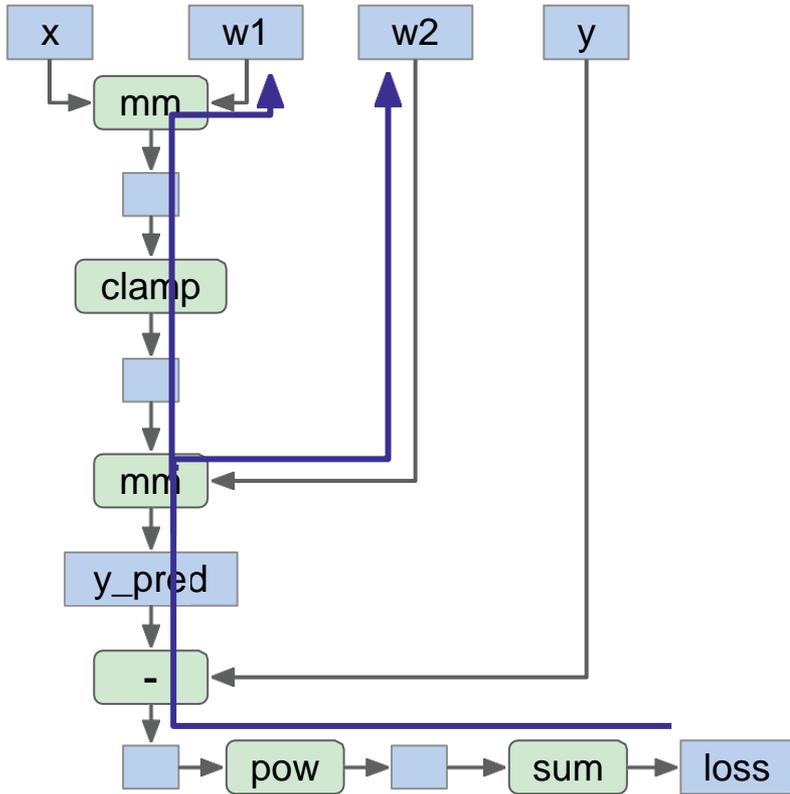
```
    y_pred = x.mm(w1).clamp(min=0).mm(w2)
```

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

```
    loss.backward()
```

Зададим структуру графа И
выполним расчет

PyTorch: Dynamic Computation Graphs



```
import torch
```

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

```
x = torch.randn(N, D_in)
```

```
y = torch.randn(N, D_out)
```

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

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

```
learning_rate = 1e-6
```

```
for t in range(500):
```

```
    y_pred = x.mm(w1).clamp(min=0).mm(w2)
```

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

```
    loss.backward()
```

Выстраивается расчет между loss и w1, w2 и выполняются вычисления

PyTorch: Dynamic Computation Graphs

**Создание графа и
вычисление по графу**
происходят одновременно.

Крайне не эффективно строить
граф 500 раз...

```
import torch

N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in)
y = torch.randn(N, D_out)
w1 = torch.randn(D_in, H, requires_grad=True)
w2 = torch.randn(H, D_out, requires_grad=True)

learning_rate = 1e-6
for t in range(500):
    y_pred = x.mm(w1).clamp(min=0).mm(w2)
    loss = (y_pred - y).pow(2).sum()

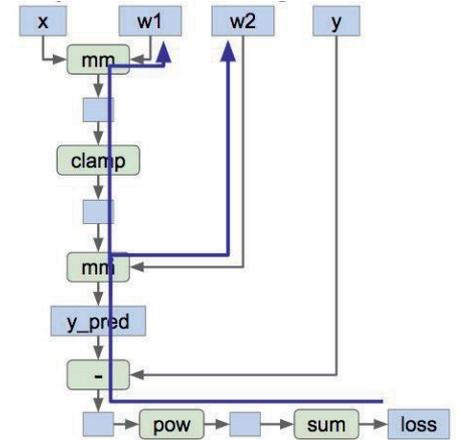
    loss.backward()
```

Static Computation Graphs

Альтернатива: **Статические** графы

Шаг 1: Построим наш граф для прямого и обратного расчета

Шаг 2: Используем этот граф на всех итерациях обучения



```
graph = build_graph()
```

```
for x_batch, y_batch in loader:  
    run_graph(graph, x=x_batch, y=y_batch)
```

TensorFlow

TensorFlow Versions

Pre-2.0 (1.14 latest)

Статические графы по-умолчанию, опционально динамические графы (eager mode).

2.1 (March 2020)

По-умолчанию динамические графы, опционально статические.

Здесь говорим про 2.1.

TensorFlow: Neural Net (Pre-2.0)

```
import numpy as np
import tensorflow as tf
```

(Это импорт, делаем
его всегда)

```
N, D, H = 64, 1000, 100
x = tf.placeholder(tf.float32, shape=(N, D))
y = tf.placeholder(tf.float32, shape=(N, D))
w1 = tf.placeholder(tf.float32, shape=(D, H))
w2 = tf.placeholder(tf.float32, shape=(H, D))

h = tf.maximum(tf.matmul(x, w1), 0)
y_pred = tf.matmul(h, w2)
diff = y_pred - y
loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))

grad_w1, grad_w2 = tf.gradients(loss, [w1, w2])

with tf.Session() as sess:
    values = {x: np.random.randn(N, D),
              w1: np.random.randn(D, H),
              w2: np.random.randn(H, D),
              y: np.random.randn(N, D),}
    out = sess.run([loss, grad_w1, grad_w2],
                    feed_dict=values)
    loss_val, grad_w1_val, grad_w2_val = out
```

TensorFlow: Neural Net (Pre-2.0)

Сначала
определим
граф



```
N, D, H = 64, 1000, 100
x = tf.placeholder(tf.float32, shape=(N, D))
y = tf.placeholder(tf.float32, shape=(N, D))
w1 = tf.placeholder(tf.float32, shape=(D, H))
w2 = tf.placeholder(tf.float32, shape=(H, D))

h = tf.maximum(tf.matmul(x, w1), 0)
y_pred = tf.matmul(h, w2)
diff = y_pred - y
loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))

grad_w1, grad_w2 = tf.gradients(loss, [w1, w2])
```

Затем **выполним**
вычисления по
графу



```
with tf.Session() as sess:
    values = {x: np.random.randn(N, D),
              w1: np.random.randn(D, H),
              w2: np.random.randn(H, D),
              y: np.random.randn(N, D),}
    out = sess.run([loss, grad_w1, grad_w2],
                    feed_dict=values)
    loss_val, grad_w1_val, grad_w2_val = out
```

TensorFlow: 2.0+ vs. pre-2.0

```
N, D, H = 64, 1000, 100

x = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
y = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
w1 = tf.Variable(tf.random.uniform((D, H))) # weights
w2 = tf.Variable(tf.random.uniform((H, D))) # weights

with tf.GradientTape() as tape:
    h = tf.maximum(tf.matmul(x, w1), 0)
    y_pred = tf.matmul(h, w2)
    diff = y_pred - y
    loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))
    gradients = tape.gradient(loss, [w1, w2])
```

Tensorflow 2.0+:

Нетерпеливые графы, “Eager”
режим по умолчанию

```
assert(tf.executing_eagerly())
```

```
N, D, H = 64, 1000, 100
x = tf.placeholder(tf.float32, shape=(N, D))
y = tf.placeholder(tf.float32, shape=(N, D))
w1 = tf.placeholder(tf.float32, shape=(D, H))
w2 = tf.placeholder(tf.float32, shape=(H, D))

h = tf.maximum(tf.matmul(x, w1), 0)
y_pred = tf.matmul(h, w2)
diff = y_pred - y
loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))

grad_w1, grad_w2 = tf.gradients(loss, [w1, w2])

with tf.Session() as sess:
    values = {x: np.random.randn(N, D),
              w1: np.random.randn(D, H),
              w2: np.random.randn(H, D),
              y: np.random.randn(N, D),}
    out = sess.run([loss, grad_w1, grad_w2],
                    feed_dict=values)
    loss_val, grad_w1_val, grad_w2_val = out
```

Tensorflow 1.13

TensorFlow: 2.0+ vs. pre-2.0

```
N, D, H = 64, 1000, 100
x = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
y = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
w1 = tf.Variable(tf.random.uniform((D, H))) # weights
w2 = tf.Variable(tf.random.uniform((H, D))) # weights
```

```
with tf.GradientTape() as tape:
    h = tf.maximum(tf.matmul(x, w1), 0)
    y_pred = tf.matmul(h, w2)
    diff = y_pred - y
    loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))
    gradients = tape.gradient(loss, [w1, w2]).
```

Tensorflow 2.0+:

Нетерпеливые графы, “Eager”
режим по умолчанию

```
assert(tf.executing_eagerly())
```

```
N, D, H = 64, 1000, 100
x = tf.placeholder(tf.float32, shape=(N, D))
y = tf.placeholder(tf.float32, shape=(N, D))
w1 = tf.placeholder(tf.float32, shape=(D, H))
w2 = tf.placeholder(tf.float32, shape=(H, D))
```

```
h = tf.maximum(tf.matmul(x, w1), 0)
y_pred = tf.matmul(h, w2)
diff = y_pred - y
loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))
```

```
grad_w1, grad_w2 = tf.gradients(loss, [w1, w2])
```

```
with tf.Session() as sess:
```

```
values = {x: np.random.randn(N, D),
          w1: np.random.randn(D, H),
          w2: np.random.randn(H, D),
          y: np.random.randn(N, D),}
```

```
out = sess.run([loss, grad_w1, grad_w2],
               feed_dict=values)
```

```
loss_val, grad_w1_val, grad_w2_val = out
```

Tensorflow 1.13

TensorFlow: 2.0+ vs. pre-2.0

```
N, D, H = 64, 1000, 100
```

```
x = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
y = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
w1 = tf.Variable(tf.random.uniform((D, H))) # weights
w2 = tf.Variable(tf.random.uniform((H, D))) # weights
```

```
with tf.GradientTape() as tape:
```

```
    h = tf.maximum(tf.matmul(x, w1), 0)
    y_pred = tf.matmul(h, w2)
    diff = y_pred - y
    loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))
    gradients = tape.gradient(loss, [w1, w2])
```

Tensorflow 2.0+:

Нетерпеливые графы, “Eager”
режим по умолчанию

```
assert(tf.executing_eagerly())
```

```
N, D, H = 64, 1000, 100
```

```
x = tf.placeholder(tf.float32, shape=(N, D))
y = tf.placeholder(tf.float32, shape=(N, D))
w1 = tf.placeholder(tf.float32, shape=(D, H))
w2 = tf.placeholder(tf.float32, shape=(H, D))
```

```
h = tf.maximum(tf.matmul(x, w1), 0)
y_pred = tf.matmul(h, w2)
diff = y_pred - y
loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))

grad_w1, grad_w2 = tf.gradients(loss, [w1, w2])
```

```
with tf.Session() as sess:
```

```
    values = {x: np.random.randn(N, D),
              w1: np.random.randn(D, H),
              w2: np.random.randn(H, D),
              y: np.random.randn(N, D),}
```

```
    out = sess.run([loss, grad_w1, grad_w2],
                    feed_dict=values)
    loss_val, grad_w1_val, grad_w2_val = out
```

Tensorflow 1.13

TensorFlow: Neural Net

Преобразуем
входные массивы из
numpy в TF **tensors**.
Веса создаем как
tf.Variable

```
N, D, H = 64, 1000, 100
```

```
x = tf.convert_to_tensor(np.random.randn(N, D), np.float32)  
y = tf.convert_to_tensor(np.random.randn(N, D), np.float32)  
w1 = tf.Variable(tf.random.uniform((D, H))) # weights  
w2 = tf.Variable(tf.random.uniform((H, D))) # weights
```

```
with tf.GradientTape() as tape:  
    h = tf.maximum(tf.matmul(x, w1), 0)  
    y_pred = tf.matmul(h, w2)  
    diff = y_pred - y  
    loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))  
gradients = tape.gradient(loss, [w1, w2].)
```

TensorFlow: Neural Net

Используем контекст
`tf.GradientTape()`
чтобы создать
динамический граф

```
N, D, H = 64, 1000, 100

x = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
y = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
w1 = tf.Variable(tf.random.uniform((D, H))) # weights
w2 = tf.Variable(tf.random.uniform((H, D))) # weights

with tf.GradientTape() as tape:
    h = tf.maximum(tf.matmul(x, w1), 0)
    y_pred = tf.matmul(h, w2)
    diff = y_pred - y
    loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))
gradients = tape.gradient(loss, [w1, w2].)
```

TensorFlow: Neural Net

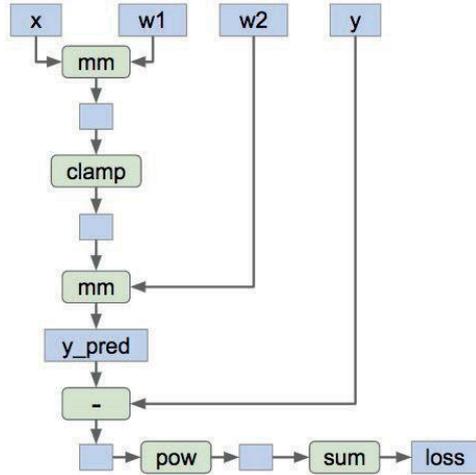
Все операции прямого расчета, включая вызовы функций, трассируются для дальнейшего расчета градиентов

```
N, D, H = 64, 1000, 100

x = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
y = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
w1 = tf.Variable(tf.random.uniform((D, H))) # weights
w2 = tf.Variable(tf.random.uniform((H, D))) # weights

with tf.GradientTape() as tape:
    h = tf.maximum(tf.matmul(x, w1), 0)
    y_pred = tf.matmul(h, w2)
    diff = y_pred - y
    loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))
gradients = tape.gradient(loss, [w1, w2].)
```

TensorFlow: Neural Net



Forward pass

```
N, D, H = 64, 1000, 100
```

```
x = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
y = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
w1 = tf.Variable(tf.random.uniform((D, H))) # weights
w2 = tf.Variable(tf.random.uniform((H, D))) # weights
```

```
with tf.GradientTape() as tape:
```

```
h = tf.maximum(tf.matmul(x, w1), 0)
y_pred = tf.matmul(h, w2)
diff = y_pred - y
loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))
gradients = tape.gradient(loss, [w1, w2].)
```

TensorFlow: Neural Net

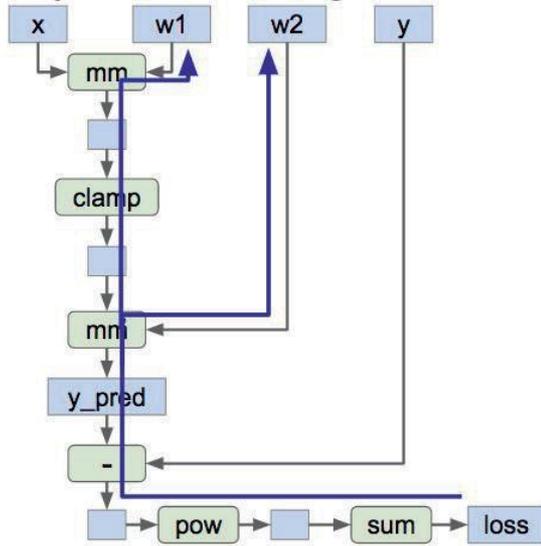
tape.gradient()
использует граф из
трассировки для
вычисления
градиентов по весам

```
N, D, H = 64, 1000, 100

x = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
y = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
w1 = tf.Variable(tf.random.uniform((D, H))) # weights
w2 = tf.Variable(tf.random.uniform((H, D))) # weights

with tf.GradientTape() as tape:
    h = tf.maximum(tf.matmul(x, w1), 0)
    y_pred = tf.matmul(h, w2)
    diff = y_pred - y
    loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))
    gradients = tape.gradient(loss, [w1, w2])
```

TensorFlow: Neural Net



Backward pass

```
N, D, H = 64, 1000, 100
```

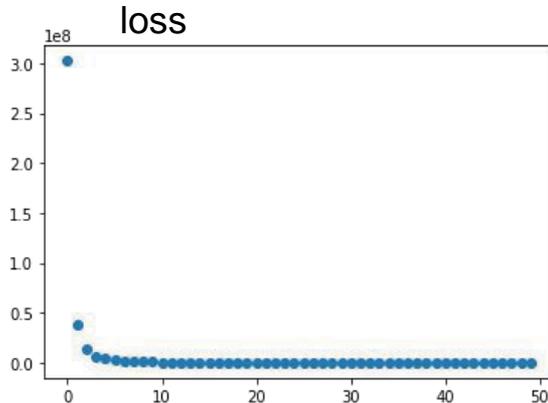
```
x = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
y = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
w1 = tf.Variable(tf.random.uniform((D, H))) # weights
w2 = tf.Variable(tf.random.uniform((H, D))) # weights
```

```
with tf.GradientTape() as tape:
    h = tf.maximum(tf.matmul(x, w1), 0)
    y_pred = tf.matmul(h, w2)
    diff = y_pred - y
    loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))
    gradients = tape.gradient(loss, [w1, w2])
```

TensorFlow: Neural Net

Обучение сети:

запустим все это в
цикле с пересчетом
весов



```
N, D, H = 64, 1000, 100
```

```
x = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
y = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
w1 = tf.Variable(tf.random.uniform((D, H))) # weights
w2 = tf.Variable(tf.random.uniform((H, D))) # weights
```

```
learning_rate = 1e-6
for t in range(50):
    with tf.GradientTape() as tape:
        h = tf.maximum(tf.matmul(x, w1), 0)
        y_pred = tf.matmul(h, w2)
        diff = y_pred - y
        loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))
        gradients = tape.gradient(loss, [w1, w2])
        w1.assign(w1 - learning_rate * gradients[0])
        w2.assign(w2 - learning_rate * gradients[1])
```

TensorFlow: Optimizer

Можно использовать
оптимизатор
(**optimizer**) для
пересчета весов
согласно градиентам

```
N, D, H = 64, 1000, 100

x = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
y = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
w1 = tf.Variable(tf.random.uniform((D, H))) # weights
w2 = tf.Variable(tf.random.uniform((H, D))) # weights

optimizer = tf.optimizers.SGD(1e-6)

learning_rate = 1e-6
for t in range(50):
    with tf.GradientTape() as tape:
        h = tf.matmul(x, w1)
        y_pred = tf.matmul(h, w2)
        diff = y_pred - y
        loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))
        gradients = tape.gradient(loss, [w1, w2])
    optimizer.apply_gradients(zip(gradients, [w1, w2])).
```

TensorFlow: Loss

Можно
использовать
типовые функции
потерь из модуля
Loss

```
N, D, H = 64, 1000, 100

x = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
y = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
w1 = tf.Variable(tf.random.uniform((D, H))) # weights
w2 = tf.Variable(tf.random.uniform((H, D))) # weights

optimizer = tf.optimizers.SGD(1e-6)

for t in range(50):
    with tf.GradientTape() as tape:
        h = tf.maximum(tf.matmul(x, w1), 0)
        y_pred = tf.matmul(h, w2)
        diff = y_pred - y
        loss = tf.losses.MeanSquaredError()(y_pred, y)
    gradients = tape.gradient(loss, [w1, w2])
    optimizer.apply_gradients(zip(gradients, [w1, w2]))
```

Keras: высокоуровневая обертка

Keras надстройка
над TensorFlow, облегчающая
типовые задачи

Keras был внешним к TF, в
2.X встроен

```
N, D, H = 64, 1000, 100

x = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
y = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(H, input_shape=(D,),
                                activation=tf.nn.relu))
model.add(tf.keras.layers.Dense(D))
optimizer = tf.optimizers.SGD(1e-1)

losses = []
for t in range(50):
    with tf.GradientTape() as tape:
        y_pred = model(x)
        loss = tf.losses.MeanSquaredError()(y_pred, y)
    gradients = tape.gradient(
        loss, model.trainable_variables)
    optimizer.apply_gradients(
        zip(gradients, model.trainable_variables))
```

Keras: высокоуровневая обертка

Определяем модель
как
последовательность
слоев

Подучаем выход
модели

Применяем
градиенты ко всем
обучаемым
параметрам модели
(весам)

```
N, D, H = 64, 1000, 100

x = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
y = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(H, input_shape=(D,),
                                activation=tf.nn.relu))
model.add(tf.keras.layers.Dense(D))
optimizer = tf.optimizers.SGD(1e-1)

losses = []
for t in range(50):
    with tf.GradientTape() as tape:
        y_pred = model(x)
        loss = tf.losses.MeanSquaredError()(y_pred, y)
        gradients = tape.gradient(
            loss, model.trainable_variables)
        optimizer.apply_gradients(
            zip(gradients, model.trainable_variables))
```

Keras: высокоуровневая обертка

```
N, D, H = 64, 1000, 100

x = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
y = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(H, input_shape=(D,),
                                activation=tf.nn.relu))
model.add(tf.keras.layers.Dense(D))
optimizer = tf.optimizers.SGD(1e-1)
model.compile(loss=tf.keras.losses.MeanSquaredError(),
              optimizer=optimizer)
history = model.fit(x, y, epochs=50, batch_size=N)
```

Keras может и весь
цикл обучения
реализовать внутри!

TensorFlow: High-Level Wrappers

Keras (<https://keras.io/>)

tf.keras (https://www.tensorflow.org/api_docs/python/tf/keras)

tf.estimator (https://www.tensorflow.org/api_docs/python/tf/estimator)

Sonnet (<https://github.com/deepmind/sonnet>)

TFLearn (<http://tflearn.org/>)

TensorLayer (<http://tensorlayer.readthedocs.io/en/latest/>)

Декоратор @tf.function: статический граф

Декоратор
tf.function (неявно)
компилирует
функцию в
статический граф
для повышения
производительности

```
N, D, H = 64, 1000, 100
x = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
y = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(H, input_shape=(D,),
                                activation=tf.nn.relu))
model.add(tf.keras.layers.Dense(D))
optimizer = tf.optimizers.SGD(1e-1)
```



```
@tf.function
def model_func(x, y):
    y_pred = model(x)
    loss = tf.losses.MeanSquaredError()(y_pred, y)
    return y_pred, loss
```

```
for t in range(50):
    with tf.GradientTape() as tape:
        y_pred, loss = model_func(x, y)
    gradients = tape.gradient(
        loss, model.trainable_variables)
    optimizer.apply_gradients(
        zip(gradients, model.trainable_variables))
```

Декоратор @tf.function: статический граф

Запускалось в Google Colab в апреле 2020

Сравниваем время
инференса для
статического и
динамического
графа

```
N, D, H = 64, 1000, 100
x = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
y = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(H, input_shape=(D,), activation=tf.nn.relu))
model.add(tf.keras.layers.Dense(D))
optimizer = tf.optimizers.SGD(1e-1)

@tf.function
def model_static(x, y):
    y_pred = model(x)
    loss = tf.losses.MeanSquaredError()(y_pred, y)
    return y_pred, loss

def model_dynamic(x, y):
    y_pred = model(x)
    loss = tf.losses.MeanSquaredError()(y_pred, y)

print("dynamic graph: ", timeit.timeit(lambda: model_dynamic(x, y), number=10))
print("static graph: ", timeit.timeit(lambda: model_static(x, y), number=10))

dynamic graph: 0.02520249200000535
static graph: 0.03932226699998864
```

Декоратор @tf.function: статический граф

Запускалось в Google Colab в апреле 2020

Теоретически,
статический граф
быстрее но все зависит
от модели, слоев, всего
графа

```
N, D, H = 64, 1000, 100
x = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
y = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(H, input_shape=(D,), activation=tf.nn.relu))
model.add(tf.keras.layers.Dense(D))
optimizer = tf.optimizers.SGD(1e-1)
```

```
@tf.function
def model_static(x, y):
    y_pred = model(x)
    loss = tf.losses.MeanSquaredError()(y_pred, y)
    return y_pred, loss
```

```
def model_dynamic(x, y):
    y_pred = model(x)
    loss = tf.losses.MeanSquaredError()(y_pred, y)

print("dynamic graph: ", timeit.timeit(lambda: model_dynamic(x, y), number=10))
print("static graph: ", timeit.timeit(lambda: model_static(x, y), number=10))
```

```
dynamic graph: 0.025202492000000535
static graph: 0.039322266999998864
```

Декоратор @tf.function: статический граф

Запускалось в Google Colab в апреле 2020

Теоретически,
статический граф
быстрее но все зависит
от модели, слоев, всего
графа

```
N, D, H = 64, 1000, 100
x = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
y = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(H, input_shape=(D,), activation=tf.nn.relu))
model.add(tf.keras.layers.Dense(D))
optimizer = tf.optimizers.SGD(1e-1)

@tf.function
def model_static(x, y):
    y_pred = model(x)
    loss = tf.losses.MeanSquaredError()(y_pred, y)
    return y_pred, loss

def model_dynamic(x, y):
    y_pred = model(x)
    loss = tf.losses.MeanSquaredError()(y_pred, y)

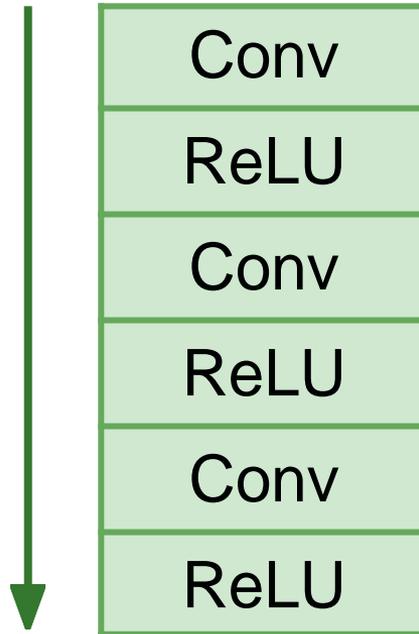
print("dynamic graph:", timeit.timeit(lambda: model_dynamic(x, y), number=1000))
print("static graph:", timeit.timeit(lambda: model_static(x, y), number=1000))

dynamic graph: 2.3648411540000325
static graph: 1.1723986679999143
```

Static vs Dynamic: Оптимизация

Статические графы лучше поддаются оптимизации

Исходный граф



Эквивалентный граф, операции объединены



Static PyTorch: Стандарт ONNX

ONNX опенсорсный стандарт обмена для нейросетевых моделей

Цель: Обучить сетку на одном фреймворке, выполнить инференс на другом

Поддерживается PyTorch, Caffe2, Microsoft CNTK, Apache MXNet

<https://github.com/onnx/onnx>

TensorFlow - <https://github.com/onnx/onnx-tensorflow>

Static PyTorch: поддержка ONNX

Можно экспортировать PyTorch модель в ONNX

Выполните модель над пустым входом чтобы создался граф, и сохраните граф

Работает если ваша модель не содержит в себе условий, циклов, т.е. не является динамическим графом

```
import torch

N, D_in, H, D_out = 64, 1000, 100, 10
model = torch.nn.Sequential(
    torch.nn.Linear(D_in, H),
    torch.nn.ReLU(),
    torch.nn.Linear(H, D_out))

dummy_input = torch.randn(N, D_in)
torch.onnx.export(model, dummy_input,
                  'model.proto',
                  verbose=True)
```

Static PyTorch: поддержка ONNX

```
graph(%0 : Float(64, 1000)
      %1 : Float(100, 1000)
      %2 : Float(100)
      %3 : Float(10, 100)
      %4 : Float(10)) {
  %5 : Float(64, 100) =
  onnx::Gemm[alpha=1, beta=1, broadcast=1,
  transB=1](%0, %1, %2), scope:
  Sequential/Linear[0]
  %6 : Float(64, 100) = onnx::Relu(%5),
  scope: Sequential/ReLU[1]
  %7 : Float(64, 10) = onnx::Gemm[alpha=1,
  beta=1, broadcast=1, transB=1](%6, %3,
  %4), scope: Sequential/Linear[2]
  return (%7);
}
```

```
import torch

N, D_in, H, D_out = 64, 1000, 100, 10
model = torch.nn.Sequential(
    torch.nn.Linear(D_in, H),
    torch.nn.ReLU(),
    torch.nn.Linear(H, D_out))

dummy_input = torch.randn(N, D_in)
torch.onnx.export(model, dummy_input,
                  'model.proto',
                  verbose=True)
```

После экспорта в ONNX,
можно запустить torch модель
например в Caffe2

Static PyTorch: TorchScript

```
graph(%self.1 :
  __torch__.torch.nn.modules.module.__torch_mangle_4.Module,
    %input : Float(3, 4),
    %h : Float(3, 4)):
  %19 :
  __torch__.torch.nn.modules.module.__torch_mangle_3.Module =
  prim::GetAttr[name="linear"](%self.1)
  %21 : Tensor =
  prim::CallMethod[name="forward"](%19, %input)
  %12 : int = prim::Constant[value=1]() #
  <ipython-input-40-26946221023e>:7:0
  %13 : Float(3, 4) = aten::add(%21, %h, %12) #
  <ipython-input-40-26946221023e>:7:0
  %14 : Float(3, 4) = aten::tanh(%13) #
  <ipython-input-40-26946221023e>:7:0
  %15 : (Float(3, 4), Float(3, 4)) =
  prim::TupleConstruct(%14, %14)
  return (%15)
```

```
class MyCell(torch.nn.Module):
    def __init__(self):
        super(MyCell, self).__init__()
        self.linear = torch.nn.Linear(4, 4)

    def forward(self, x, h):
        new_h = torch.tanh(self.linear(x) + h)
        return new_h, new_h

my_cell = MyCell()
x, h = torch.rand(3, 4), torch.rand(3, 4)
traced_cell = torch.jit.trace(my_cell, (x, h))
print(traced_cell.graph)
traced_cell(x, h)
```

Экспорт сериализованного графа в TorchScript позволит запустить его без питон-окружения, в Java, C++, JS...

PyTorch vs TensorFlow, Static vs Dynamic

PyTorch

Динамические графы

Если нужно

статические:

ONNX,

Caffe2, TorchScript

TensorFlow 2.X

Динамические: Eager

Статические: @tf.function

Static vs Dynamic: сериализация

Static

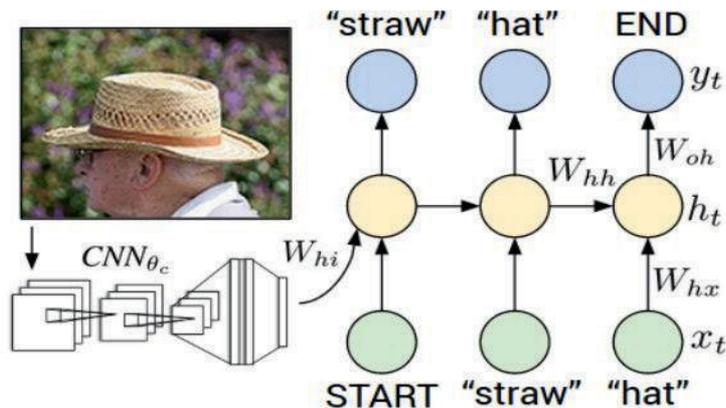
Построенный граф
можно **сериализовать**
и потом запускать, без
дополнительного кода!

Dynamic

Построение и исполнение
кода перемешаны, поэтому
код всегда нужен

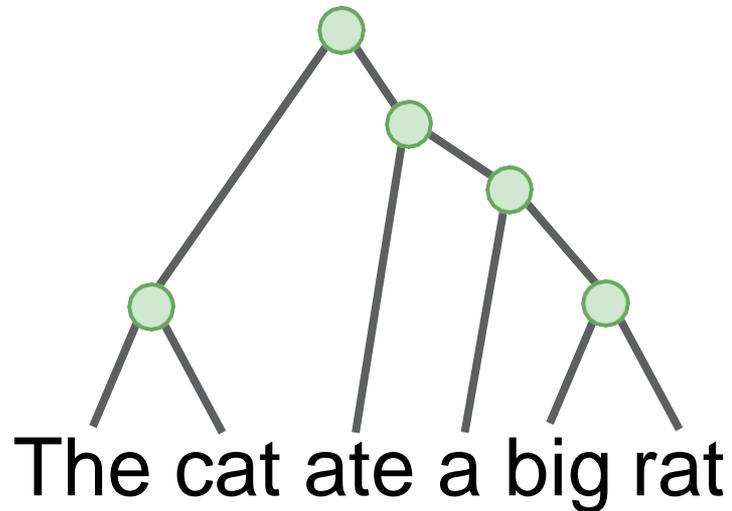
Применение динамических графов

- Рекуррентные сети



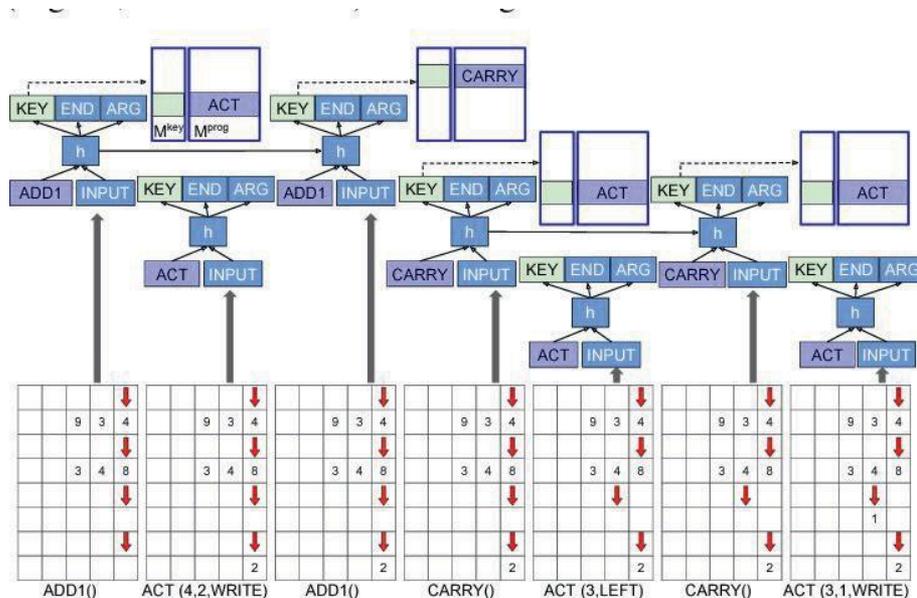
Применение динамических графов

- Рекуррентные сети
- Рекурсивные сети



Применение динамических графов

- Рекуррентные сети
- Рекурсивные сети
- Модульные сети
- Нейросетевое программирование



Применение динамических графов

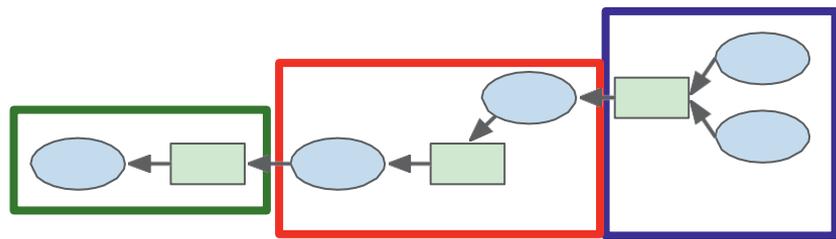
- Рекуррентные сети
- Рекурсивные сети
- Модульные сети
- Нейросетевое программирование
- Что еще???

Model Parallel vs. Data Parallel

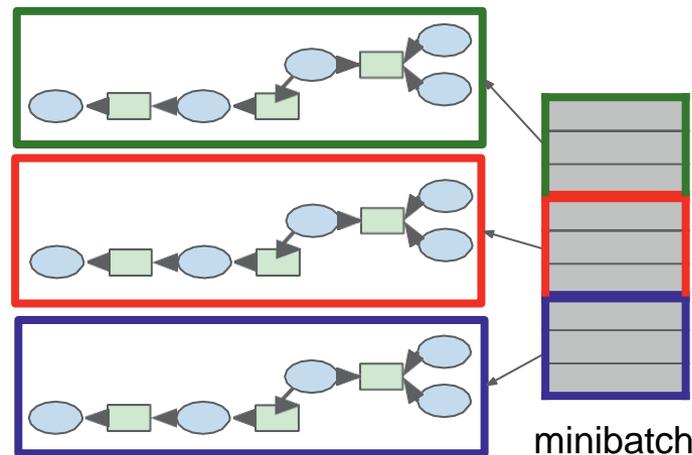
Model parallel:
разобьем граф на
части по разным
GPU



Data parallel: разобьем
батч на части по
разным GPU



Model Parallel



Data Parallel

PyTorch: Data Parallel

`nn.DataParallel`

++: очень просто использовать, просто завернуть модель в это

--: Работает внутри одного ЦПУ процесса и хоста, ЦПУ станет проблемой при большом количестве GPU (8+).

`nn.DistributedDataParallel`

++: Мультипроцессное и мультихостовое обучение

--: Настраивается вручную, скрипты запускаются вручную

Horovod (<https://github.com/horovod/horovod>): Поддерживает и PyTorch и TensorFlow

<https://pytorch.org/docs/stable/nn.html#dataparallel-layers-multi-gpu-distributed>

TensorFlow: Data Parallel

`tf.distributed.Strategy`

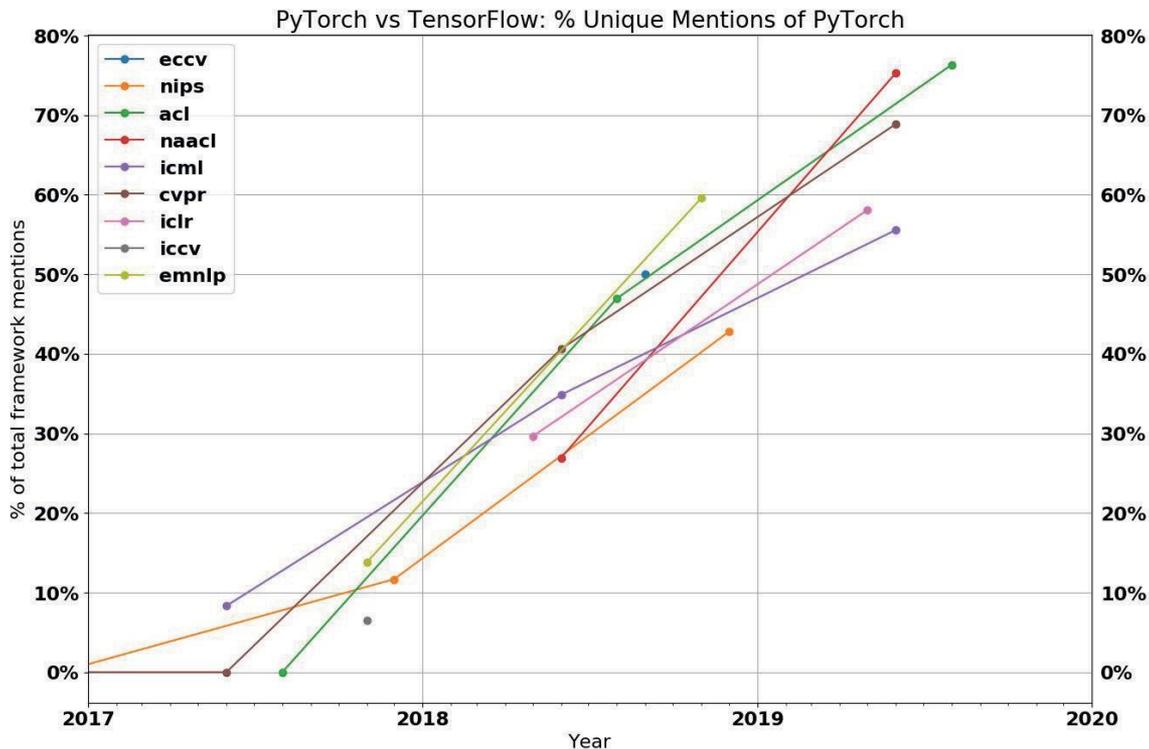
```
strategy = tf.distribute.MirroredStrategy()

with strategy.scope():
    model = tf.keras.Sequential([
        tf.keras.layers.Conv2D(32, 3, activation='relu', input_shape=(28, 28, 1)),
        tf.keras.layers.MaxPooling2D(),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(64, activation='relu'),
        tf.keras.layers.Dense(10)
    ])

    model.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                  optimizer=tf.keras.optimizers.Adam(),
                  metrics=['accuracy'])
```

<https://www.tensorflow.org/tutorials/distribute/keras>

PyTorch vs. TensorFlow: Academia



<https://thegradiant.pub/state-of-ml-frameworks-2019-pytorch-dominates-research-tensorflow-dominates-industry/>

PyTorch vs. TensorFlow: Academia

CONFERENCE	PT 2018	PT 2019	PT GROWTH	TF 2018	TF 2019	TF GROWTH
CVPR	82	280	240%	116	125	7.7%
NAACL	12	66	450%	34	21	-38.2%
ACL	26	103	296%	34	33	-2.9%
ICLR	24	70	192%	54	53	-1.9%
ICML	23	69	200%	40	53	32.5%

<https://thegradients.pub/state-of-ml-frameworks-2019-pytorch-dominates-research-tensorflow-dominates-industry/>

PyTorch vs. TensorFlow: Industry

- Официального сравнения нет.
- Требуются позиции 2389 для TensorFlow и 1366 для PyTorch.
- Перспективы не ясные, индустрия не очень рада уходить с TF
- TensorFlow все еще доминирует для мобильных или встраиваемых устройств, для high performance inference

Что выбрать:

PyTorch – хорош для исследователей, чистое API, хорошие динамические графы удобно отлаживать. Статический граф можно получить при помощи TrchScript.

TensorFlow выбор по умолчанию как минимум в индустрии. Синтаксис стал сильно лучше начиная с 2.0. Не идеален, но очень большое сообщество и много приложений. Один фреймворк для исследований и для прода. Высокоуровневые обертки вроде Keras – очень хороши.

Далее:

Обучение нейронных сетей