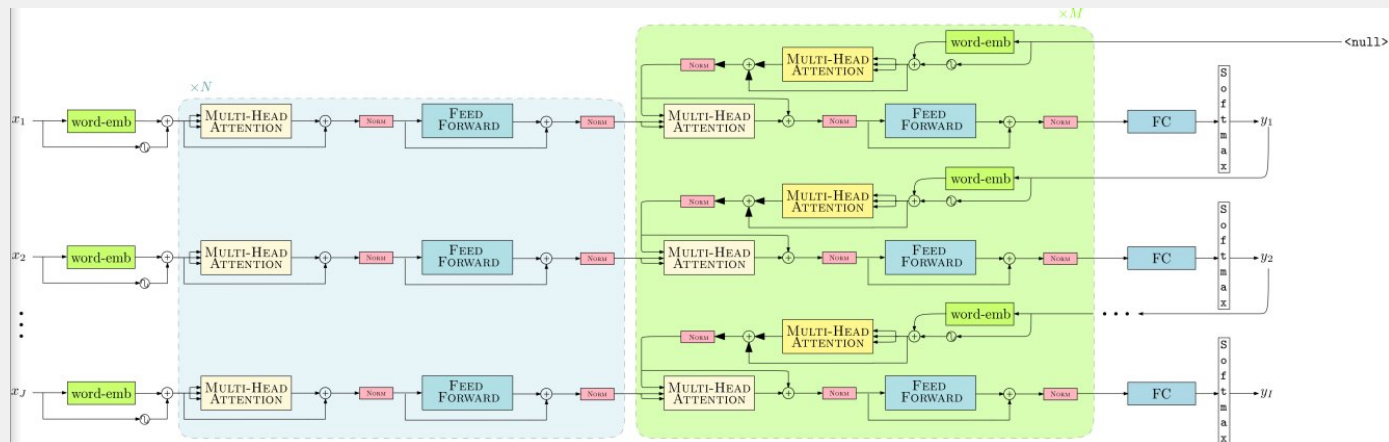
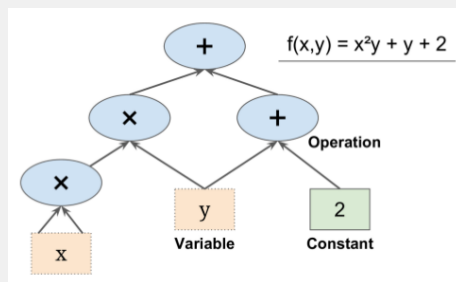


ML FRAMEWORKS

1. COMPUTATION GRAPH AND TENSORS

Модель в глубоком обучении — это композиция множества нелинейных преобразований (функций), которая также называется *поток* или *графом вычислений*.



1.1. Тензоры.

На графе вычислений *ребра* задают поток данных (тензоров). А *вершины* — операции над этими тензорами.

Таким образом, данные представляются в виде *тензоров*.

Опр. 1. (алгебра) *Тензор* (tensor) — это полилинейная функция $f : V^* \times \dots \times V^* \times V \times \dots \times V \rightarrow \mathbb{R}$.

Опр. 2. (классич.) *Тензор* — это геометрический объект, который описывается многомерным массивом, то есть набором чисел (компонент), занумерованных несколькими индексами, причем задан закон преобразования компонент при замене базиса.

В машинном обучении тензор = многомерный массив однородных данных.

Тензор задается рангом и формой:

- *ранг* (rang) — количество измерений.
- *форма* (shape) — размер (количество компонент) по каждому измерению.

Пример, матрица 3×4 — тензор с rang=2 и shape=[3,4].

2. Библиотеки глубокого обучения

2.1. Зачем нужны фреймворки DL?.

- упрощение описания нейросетевой модели;
- упрощение ее обучения за счет автоматического дифференцирования;
- ускорение обучения за счет использования GPU / TPU;
- поддержка распределенных вычислений;

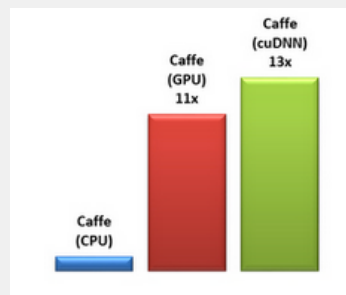


Рис. 1. Caffe and GPU TitanZ

CUDA — программно-аппаратная архитектура для выполнения параллельных вычислений на графических процессорах фирмы Nvidia.

cuDNN (CUDA Deep Neural Network) — библиотека для ускорения обучения глубоких нейронных сетей на GPU / CUDA.

Google TPU (Google Tensor Processing Unit) — тензорный процессор, разработанной компанией Google и предназначенной для обучения нейронных сетей с помощью библиотеки TensorFlow.



Рис. 2. Cloud TPU beta

- производительность: 180 терафлоп.
- стоимость вычислительного времени: \$6.50/час.

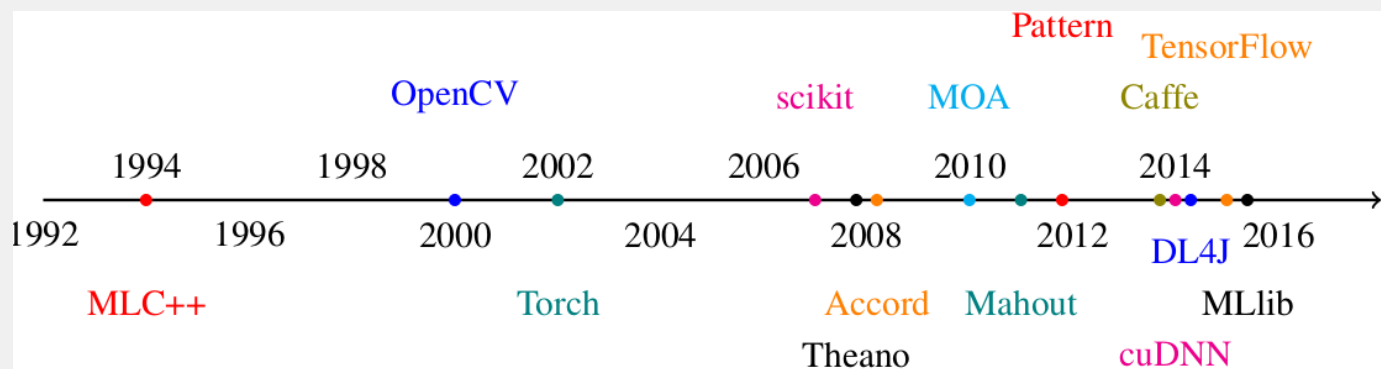
2.2. Фреймворки для ML/DL.

Любой фреймворк DL должен уметь три вещи:

- Определять граф вычислений;
- Дифференцировать граф вычислений;
- Вычислять его.

Три категории ML-фреймворков:

- *Фиксированные модули*: Caffe, Caffe2, CNTK, Kaldi, DL4J, Keras (как интерфейс): пользователь комбинирует заранее определенные блоки в граф вычислений и запускает его.
- *Статический граф вычислений*: **Theano**, **TensorFlow**, **MXNet** — граф вычислений описывается на декларативном языке, который затем компилируется в некоторый внутренний байткод, и становится твердым и монолитным. В момент компиляции граф может быть продифференцирован (например, символично). После компиляции граф может только исполняться (в прямом или обратном направлениях).
- *Динамический граф вычислений*: Torch и PyTorch — граф строится динамически каждый раз при прямом проходе, и может исполняться в прямом и обратном направлениях.



Фреймворк	Разработчик	Год выхода	Язык интерфейса
Torch	Р. Коллобер и др.	2002	Lua
Theano	Universite de Montreal	2007	Python
Caffe	Berkeley Vision	2014	Python, C++, MATLAB
TensorFlow	Google	2015	Python, C++, Go и др.
Keras	Ф. Шолле и др.	2015	Python, R
Chainer	IBM, Intel и др.	2015	Python
PyTorch	Facebook	2016	Python
CNTK	Microsoft	2016	C++
MXNet	Apache	2016	Python, R, C++, Scala и др.
DL4J	Skyminde engineering team	2016	Java, Scala и др.

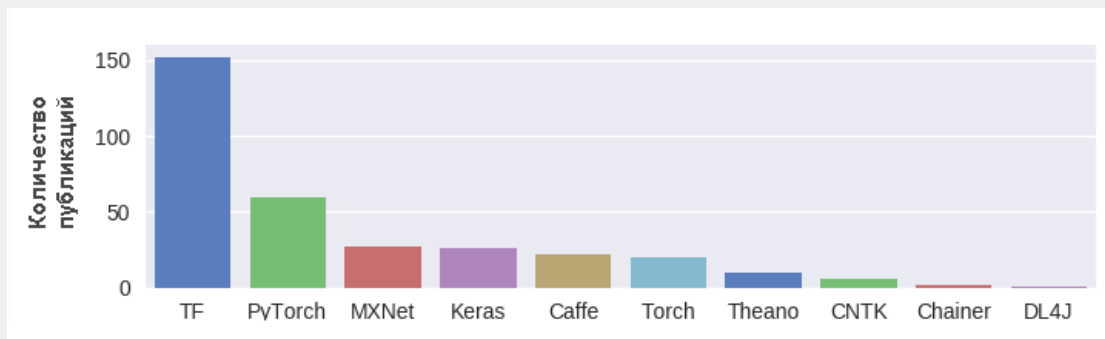


Рис. 3. Количество препринтов на arXiv.org за 2018 год

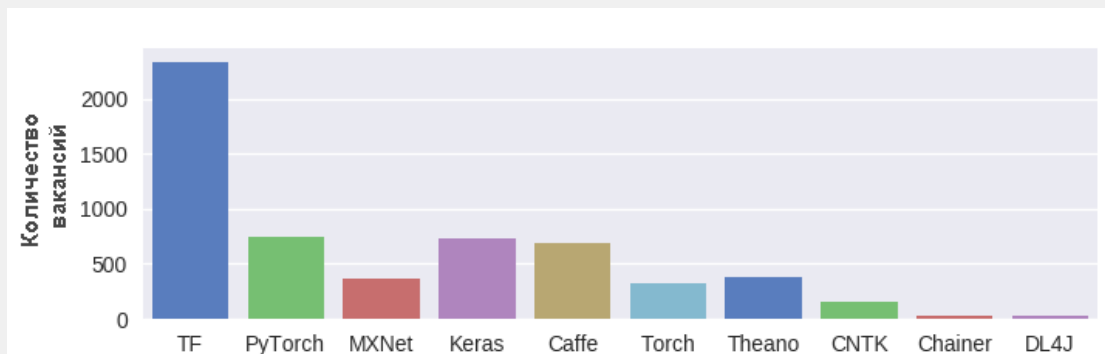


Рис. 4. Количество вакансий на indeed.com (январь 2019)

2.3. Пример описания модели на TF.

```
1 x = tf.placeholder(tf.float32, [None, 784]) # Placeholder for input.
2 y = tf.placeholder(tf.float32, [None, 10])  # Placeholder for labels.
3
4 W1 = tf.Variable(tf.random_uniform([784, 100])) # 784x100 weight matrix.
5 b1 = tf.Variable(tf.zeros([100]))              # 100-element bias vector.
6 layer1 = tf.nn.relu(tf.matmul(x, W1) + b1)     # Output of hidden layer.
7
8 W2 = tf.Variable(tf.random_uniform([100, 10])) # 100x10 weight matrix.
9 b2 = tf.Variable(tf.zeros([10]))              # 10-element bias vector.
10 layer2 = tf.matmul(layer1, W2) + b2          # Output of linear layer.
11
12 logits = layer2
```

Keras — написанная на Python открытая нейросетевая библиотека, которая является надстройкой над фреймворками TensorFlow, Theano, Deeplearning4j и др.

```
1 input_shape = (img_rows, img_cols, 1)
2 model = Sequential()
3 model.add(Conv2D(32, kernel_size=(3, 3), activation='relu',
4                 input_shape=input_shape))
5 model.add(Conv2D(64, (3, 3), activation='relu'))
6 model.add(MaxPooling2D(pool_size=(2, 2)))
7 model.add(Dropout(0.25))
8 model.add(Flatten())
9 model.add(Dense(128, activation='relu'))
10 model.add(Dropout(0.5))
11 model.add(Dense(num_classes, activation='softmax'))
12
13 model.compile(loss=keras.losses.categorical_crossentropy,
14               optimizer=keras.optimizers.Adadelta(),
15               metrics=['accuracy'])
```

PyTorch — аналог фреймворка глубокого обучения Torch7 для языка Python. Разработка PyTorch началась в Facebook в 2012 г., но открытым и доступным он стал лишь в 2017 г., а в 2018 г. вышла версия 1.0.

В PyTorch граф строится *динамически* одновременно с его вычислением. С помощью автоматического дифференцирования (automatic differentiation, AD) граф может быть продифференцирован в любой момент времени в любом его состоянии.

PyTorch = NumPy + CUDA + AD

Пример кода:

```
1 from __future__ import print_function
2 import torch
3 x = torch.rand(5, 3)
4 print(x) # !!!
```

3. FIRST STEPS: TENSORS IN PYTORCH

`torch.Tensor` — класс, реализующий многомерные массивы данных одного типа.

3.1. Создание тензоров.

```
1  import torch
2  x = torch.zeros(5, 3, dtype=torch.long) # set data type
3  print(x.size())
4
5  # Create a tensors based on an existing tensor:
6  x = x.new_ones(5, 3, dtype=torch.double)
7  x = torch.randn_like(x, dtype=torch.float)
8
9  # Create tensors from existing data:
10 x = torch.tensor([6.0, 4.5, -2.4])
11 arr = np.ones([3,3])
12 x = torch.tensor(arr)          # copies
13 x = torch.from_numpy(arr)      # doesn't copy
14 x = torch.as_tensor(arr)       # doesn't copy
15 b = x.numpy()                  # convert to numpy-array (doesn't copy):
```

```
1 x = torch.IntTensor(3, 4)
2 x.zero_()
3 x.fill_(3)
```

3.2. Типы данных (dtype).

- torch.float16 or torch.half
- torch.float32 or torch.float
- torch.float64 or torch.double
- torch.uint8
- torch.int8
- torch.int16 or torch.short
- torch.int32 or torch.int
- torch.int64 or torch.long

3.3. Операции над тензорами.

1) Способы сложения тензоров:

```
1 x = torch.rand(3, 3)
2 y = torch.rand(3, 3)
3 z = x + y
4 z = torch.add(x, y)
5 x.add_(y)  # x = x + y
```

(Методы вида `method_` изменяют сам объект — «inplace»)

2) Индексация и срезы — как в numpy:

```
1 print(x[:, 1])
```

3) GPU \longleftrightarrow CPU:

```
1 y = x.cuda()           # copy x into CUDA memory
2 y.get_device()         # return the device ordinal of GPU
3 x = y.cpu()            # copy y into CPU memory
4 y = x.to(device)       # move onto any device
```

Некоторые операции:

- `x.t()`, `x.t_()` — транспонирование;
- `x * y` — поэлементное умножение;
- `x.mm(y)`, `torch.mm(x, y)` — матричное умножение;
- `x.mv(v)`, `torch.mv(x, v)` — умножение матрицы на вектор;
- `x.dot(y)`, `torch.dot(x, y)` — скалярное умножение тензоров;

3.4. Устройства.

`watch -n 1 nvidia-smi` — NVIDIA System Management Interface

```
1 import torch
2 torch.cuda.is_available()
3 torch.cuda.device_count()
4 torch.cuda.current_device()
5 torch.cuda.get_device_name(0)
6 #torch.cuda.device(0)
```

```
1 cuda0 = torch.device('cuda:0')
2 cpu = torch.device('cpu')
```


3.5. Переменные (Variables).

Переменные — экземпляры класса `torch.autograd.Variable` — обертки над тензорами (экземплярами класса `Tensor`).

```
1 import torch
2 from torch.autograd import Variable
3 t = torch.randn(3,3)
4 a = Variable(t, requires_grad=True)
5 print(a)
6 print(a.data)
```

3.6. torch.autograd.

```
1 import torch
2 from torch.autograd import Variable
3 #t = torch.randn(3,3)
4 a = Variable(torch.Tensor([3]), requires_grad=True)
5 b = Variable(torch.Tensor([4]), requires_grad=True)
6 z = 2*a*b + 5*b
7 print(z.backward())
8 print(a.grad)
9 print(b.grad)
```

`backward()` — вычисляет градиент для текущего тензора, используя цепное правило.

4. СТРОИТЕЛЬНЫЕ БЛОКИ НЕЙРОННЫХ СЕТЕЙ



1) Полносвязный слой: $y = f(Wx + b)$

Линейное преобразование входных данных: $z = Wx + b$

```
1 nn.Linear(in_features, out_features, bias=True)
```

2) Сверточный слой и слой подвыборки:

```
1 nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, bias=True)
2 nn.functional.max_pool2d(input, kernel_size, stride=None, padding=0)
```

3) Dropout — метод регуляризации, который заключается в случайном занулении элементов (нейронов) слоя.

```
1 nn.Dropout2d(p=0.5, inplace=False)
```

p — вероятность, что элемент будет обнулен (to be zeroed).

4) BatchNorm — нормализация пакета данных, которая позволяет ускорить обучение.

```
1 nn.BatchNorm2d(num_features)
```

Пример нейросетевой модели:

https://pytorch.org/tutorials/beginner/blitz/neural_networks_tutorial.html

5. TORCHVISION

`torchvision` — это набор готовых датасетов и моделей.

```
1 root = '/data/dataset'
2 import torchvision.datasets as dset
3 dataset = dset.ImageFolder(root=root)
4 import torch.utils.data as data
5 ds = data.DataLoader(dataset)
```

5.1. **Install.** 1) 1-ый способ:

```
sudo pip3 install pytorch
```

(будет перенаправление на офиц. сайт pytorch)

2) 2-ой сп-б — самому зайти на <https://pytorch.org/>.

```
pip3 install http://download.pytorch.org/whl/cpu/torch-0.4.1-cp35-cp35m-linux_x86_64.whl
```

```
sudo pip3 install torchvision
```

`torchvision` — это набор готовых датасетов и моделей.

6. REFERENCE

<https://heartbeat.fritz.ai/introduction-to-pytorch-for-deep-learning-5b437cea90ac> —
Introduction to PyTorch for Deep Learning