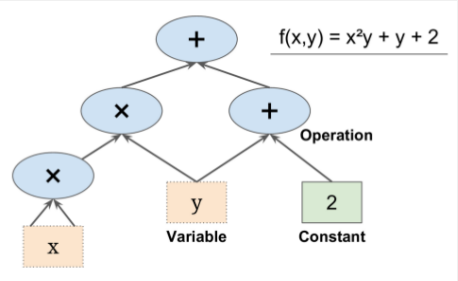
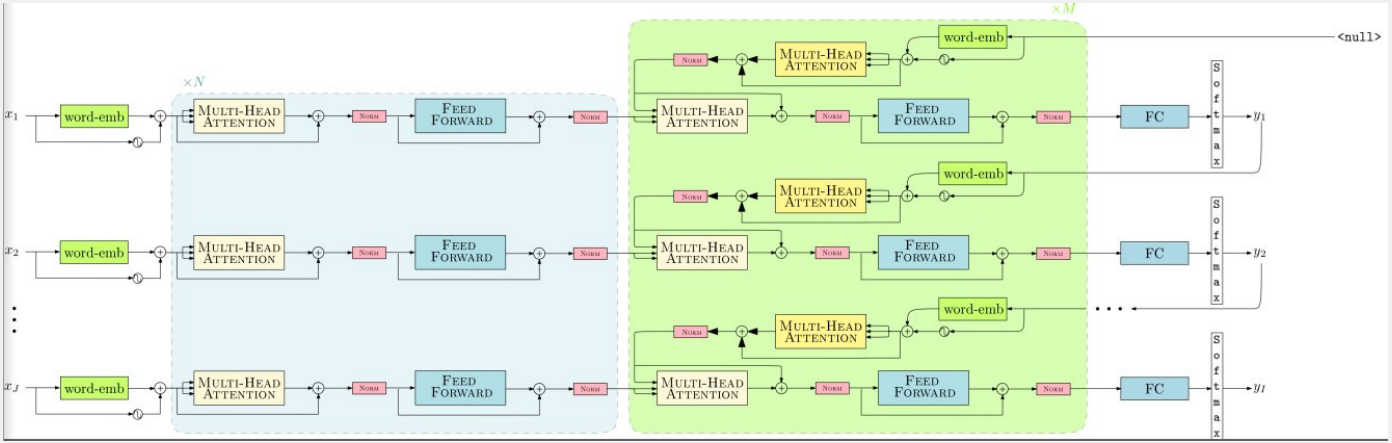
ML FRAMEWORKS

1. Computation graph and tensors

Модель в глубоком обучении — это композиция множества нелинейных преобразований (функций), которая также называется *потоком* 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 $f : V^* \times ... \times V^* \times V \times ... \times V \to \mathbb{R}$.

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 формом:

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

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

2.1. **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;

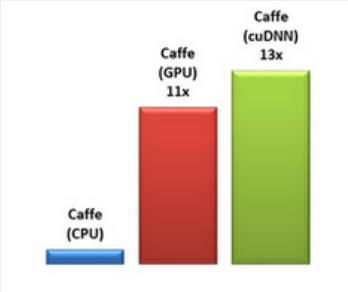


Рис. 1. Caffe and GPU TitanZ

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.

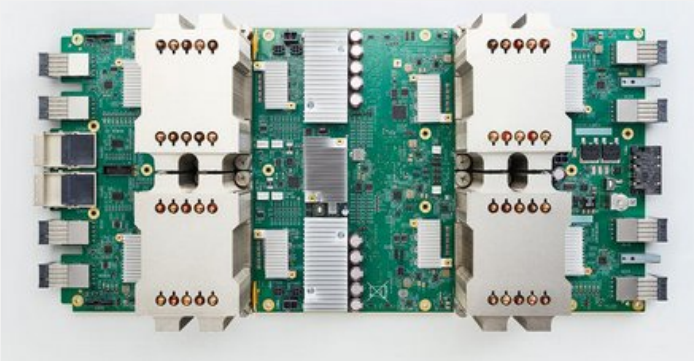


Рис. 2. Cloud TPU beta

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

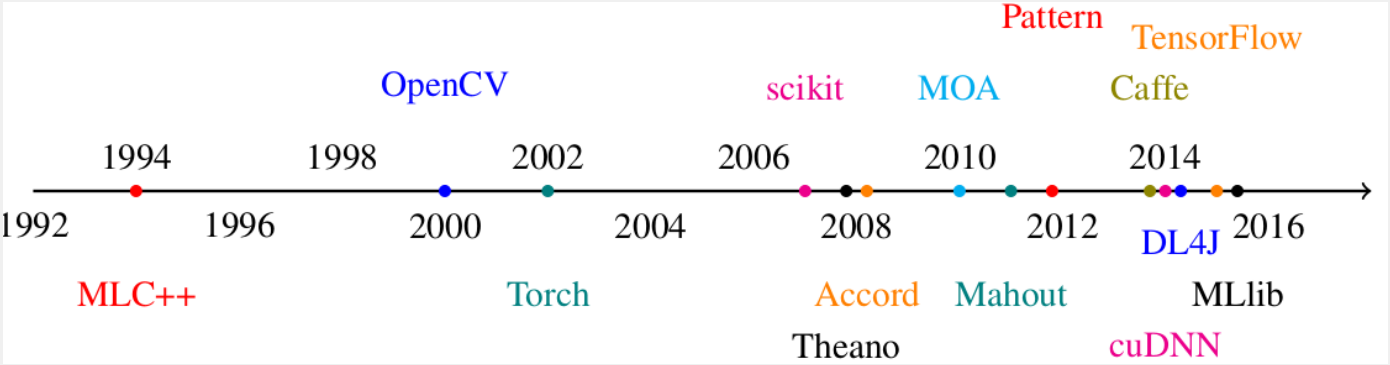
2.2. **Фреймворки для ML/DL.**

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 описавается на декларативном языке, который затем компилируется в некоторый внутренний байткод, и становится твердым и монолитным. В момент компиляции граф может быть продифференцирован (например, символьно). 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	Skymind engineering team	2016	Java, Scala и др.

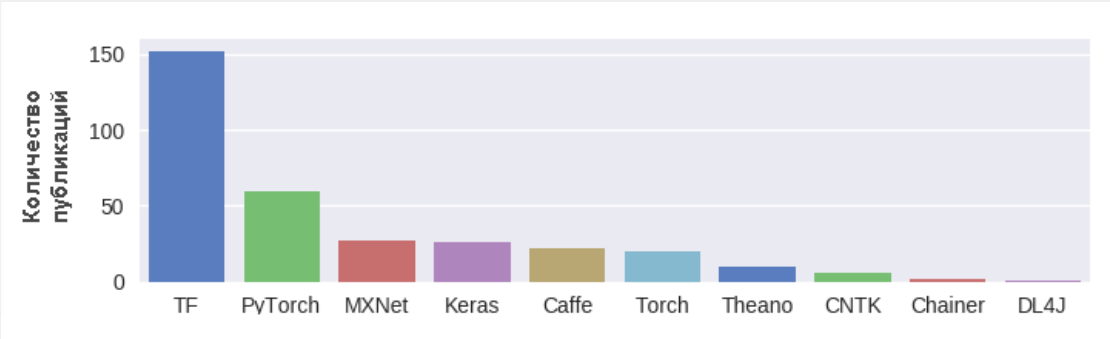


Рис. 3. Количество препринтов на arXiv.org за 2018 год

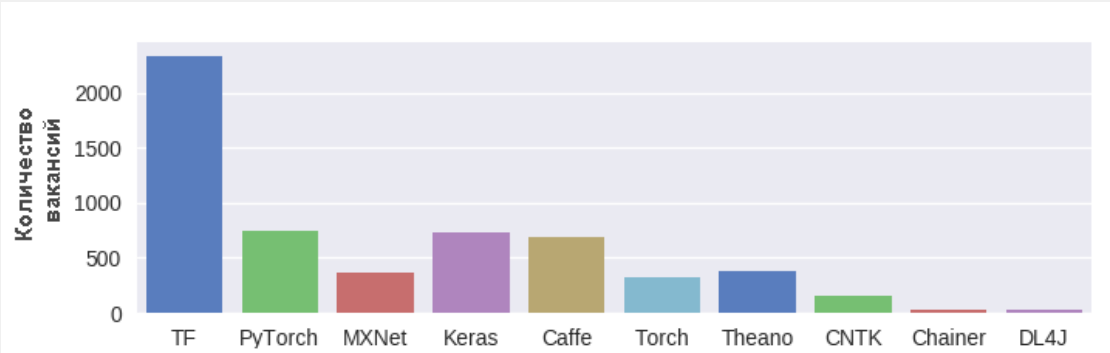


Рис. 4. Количество вакансий на indeed.com (янв 2019)

2.3. Пример описания модели на TF.

```
x = tf.placeholder(tf.float32, [None, 784]) # Placeholder for input.
y = tf.placeholder(tf.float32, [None, 10])  # Placeholder for labels.

W1 = tf.Variable(tf.random_uniform([784, 100])) # 784x100 weight matrix.
b1 = tf.Variable(tf.zeros([100]))             # 100-element bias vector.
layer1 = tf.nn.relu(tf.matmul(x, W1) + b1)    # Output of hidden layer.

W2 = tf.Variable(tf.random_uniform([100, 10]))   # 100x10 weight matrix.
b2 = tf.Variable(tf.zeros([10]))              # 10-element bias vector.
layer2 = tf.matmul(layer1, W2) + b2       # Output of linear layer.

logits = layer2
```

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

```
input_shape = (img_rows, img_cols, 1)
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
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) # !!!
```

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

3.1. Создание тензоров.

```
import torch
x = torch.zeros(5, 3, dtype=torch.long) # set data type
print(x.size())

# Create a tensors based on an existing tensor:
x = x.new_ones(5, 3, dtype=torch.double)
x = torch.randn_like(x, dtype=torch.float)

# Create tensors from existing data:
x = torch.tensor([6.0, 4.5, -2.4])
arr = np.ones([3,3])
x = torch.tensor(arr)      # copies
x = torch.from_numpy(arr)  # doesn't copy
x = torch.as_tensor(arr)   # doesn't copy
b = x.numpy()              # convert to numpy-array (doesn't copy):
```

```
x = torch.IntTensor(3, 4)
x.zero_()
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:

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

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

2) Индексация и срезы — как в numpy:

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

3) GPU $\longleftrightarrow$ CPU:

```
y = x.cuda()         # copy x into CUDA memory
y.get_device()       # return the device ordinal of GPU
x = y.cpu()          # copy y into CPU memory
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

```
import torch
torch.cuda.is_available()
torch.cuda.device_count()
torch.cuda.current_device()
torch.cuda.get_device_name(0)
#torch.cuda.device(0)
```

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

3.5. **Переменные (Variables).**

Переменные — экземпляры класса `tourch.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.

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

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

4. Слои нейронных сетей

1) Полносвязный слой: $y = f(Wx + b)$

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

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

2) Сверточный слой и слой подвыборки:

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

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

```
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.

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

`torchvision` — это набор готовых датасетов и моделей.

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

Предобученные модели:

- `resnet18, resnet34, resnet50, resnet101, resnet152`
- `inception_v3`
- `vgg11, vgg13, vgg16, vgg19, vgg11_bn. vgg13_bn, vgg16_bn, vgg19_bn`
- `alexnet`
- `squeezenet1_0, squeezenet1_1`
- `densenet121, densenet169, Densenet201`

6. Optimazers

```
optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.0005, momentum=0.9)
optimizer_ft = optim.Adam(model_ft.parameters(), lr=0.001)
```

Основные оптимизаторы:

- Adagrad
- Adam
- RMSprop

7. Layers

Заморозка слоев

```
ct = 0
for name, child in model_ft.named_children():
   print("{}: {}".format(ct, name), end='')
   ct += 1
   if ct < 7:
      for name2, params in child.named_parameters():
         params.requires_grad = False
   else:
```