Deep neural networks

AI TECH - Akademia Innowacyjnych Zastosowań Technologii Cyfrowych. Programu Operacyjnego Polska Cyfrowa na lata 2014-2020

Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Rozwoju Regionalnego
Program Operacyjny Polska Cyfrowa na lata 2014-2020, Oś Priorytetowa nr 3 "Cyfrowe kompetencje społeczeństwa"
Działanie nr 3.2 "Innowacyjne rozwiązania na rzecz aktywizacji cyfrowej"
Tytuł projektu: „Akademia Innowacyjnych Zastosowań Technologii Cyfrowych (AI Tech)"

Frameworks

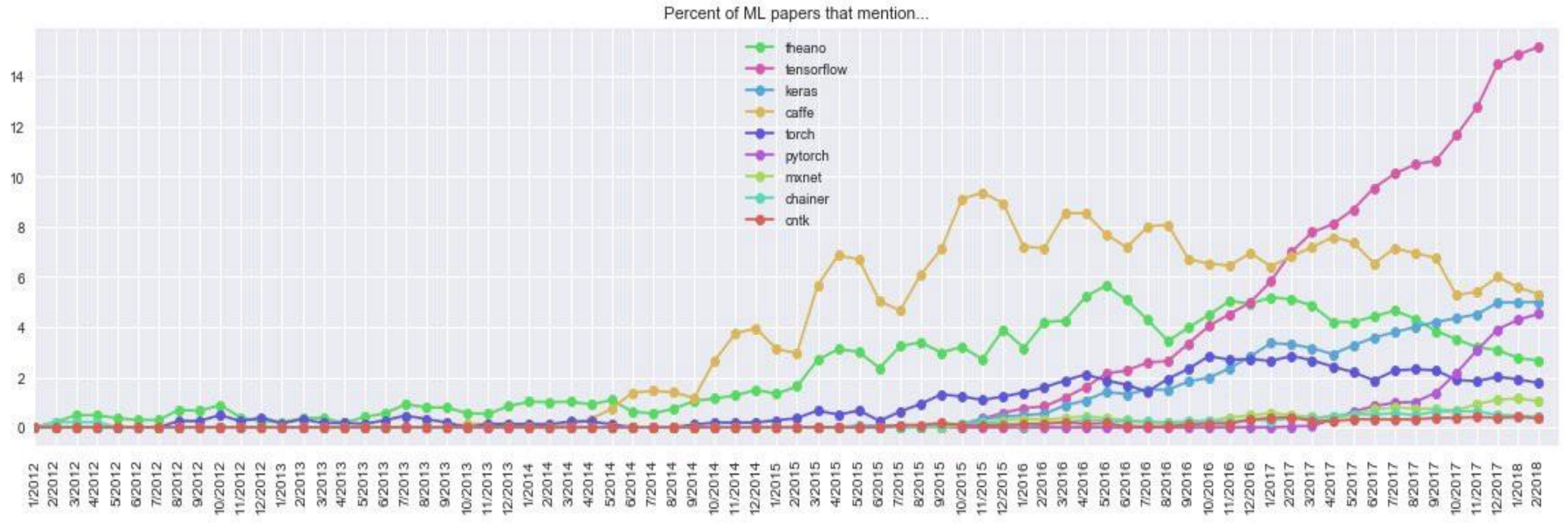
Why are they useful?

- Automatic gradient computation.

- Taking advantage of hardware (GPUs, TPUs, embedded, etc.)

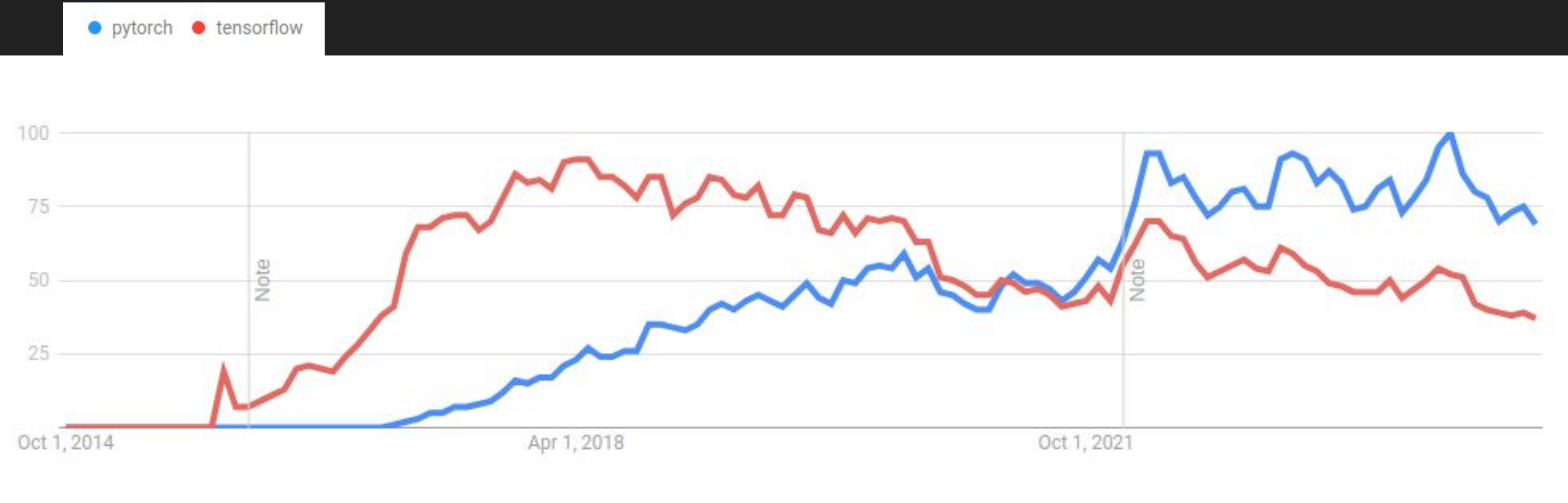
- Standardization enabling shared models (e.g. for transfer learning)

- Faster development:
    - High level interfaces.
    - Visualization, plots, etc.

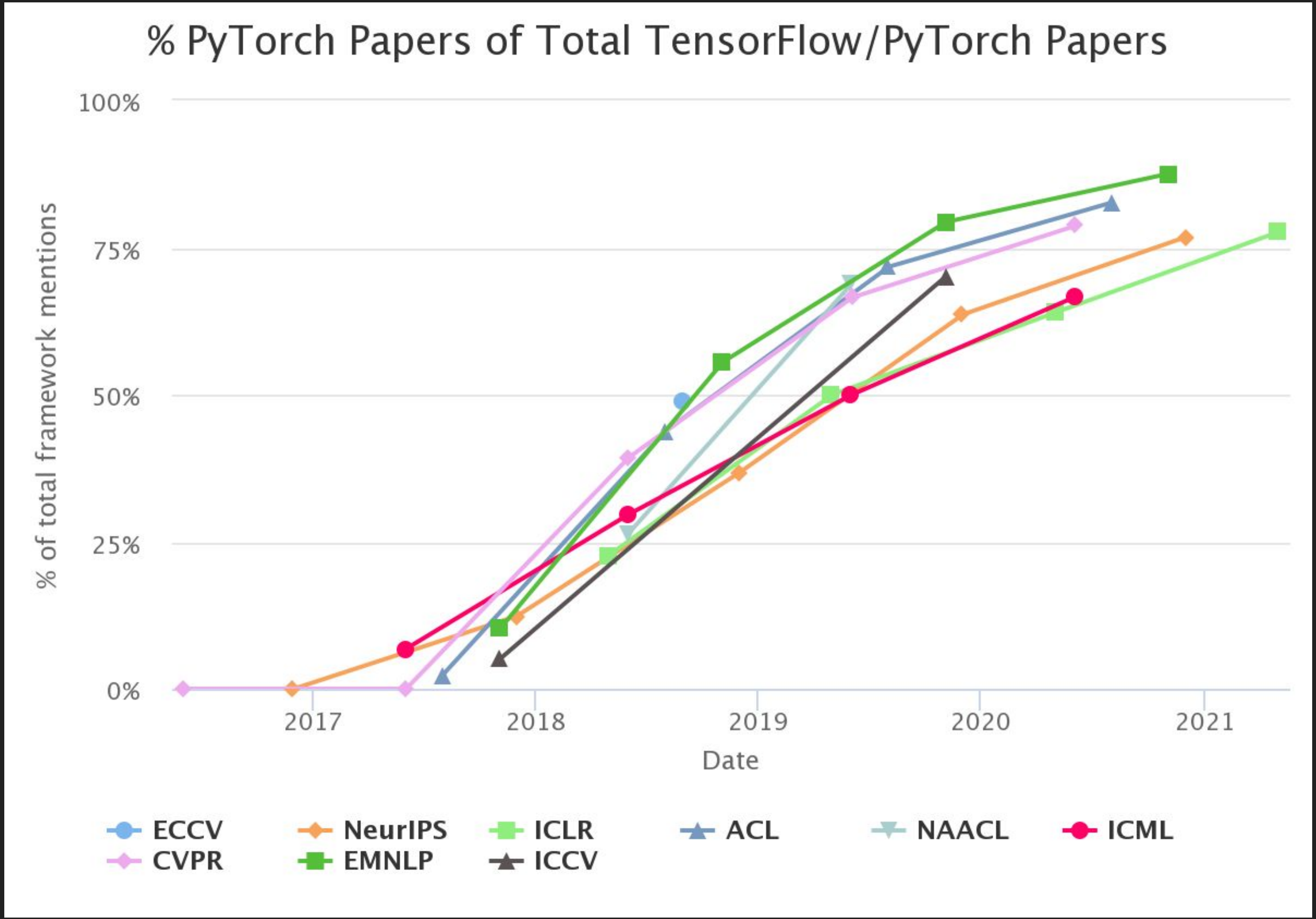

Percent of ML papers that mention...
theano
tensorflow
keras
caffe
torch
pytorch
mxnet
chainer
cntk
14
12
10
8
6
4
2
0
1/2012
2/2012
3/2012
4/2012
5/2012
6/2012
7/2012
8/2012
9/2012
10/2012
11/2012
12/2012
1/2013
2/2013
3/2013
4/2013
5/2013
6/2013
7/2013
8/2013
9/2013
10/2013
11/2013
12/2013
1/2014
2/2014
3/2014
4/2014
5/2014
6/2014
7/2014
8/2014
9/2014
10/2014
11/2014
12/2014
1/2015
2/2015
3/2015
4/2015
5/2015
6/2015
7/2015
8/2015
9/2015
10/2015
11/2015
12/2015
1/2016
2/2016
3/2016
4/2016
5/2016
6/2016
7/2016
8/2016
9/2016
10/2016
11/2016
12/2016
1/2017
2/2017
3/2017
4/2017
5/2017
6/2017
7/2017
8/2017
9/2017
10/2017
11/2017
12/2017
1/2018
2/2018

[Source]

Google trends (worldwide)

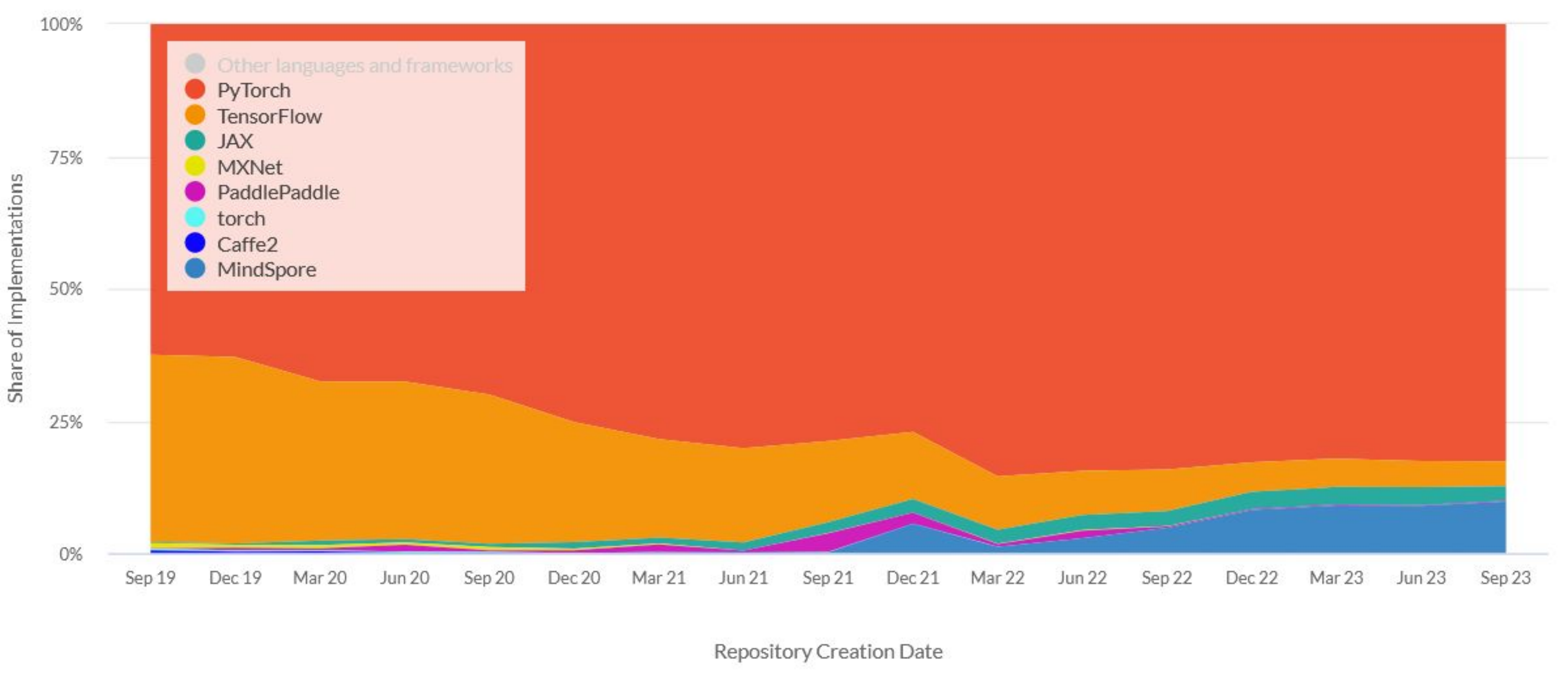


Source



% PyTorch Papers of Total TensorFlow/PyTorch Papers
100%
75%
50%
25%
0%
% of total framework mentions
2017
2018
2019
2020
2021
Date
ECCV
NeurIPS
ICLR
ACL
NAACL
ICML
CVPR
EMNLP
ICCV

Source



Other languages and frameworks
PyTorch
TensorFlow
JAX
MXNet
PaddlePaddle
torch
Caffe2
MindSpore
100%
75%
50%
25%
0%
Share of Implementations
Sep 19
Dec 19
Mar 20
Jun 20
Sep 20
Dec 20
Mar 21
Jun 21
Sep 21
Dec 21
Mar 22
Jun 22
Sep 22
Dec 22
Mar 23
Jun 23
Sep 23
Repository Creation Date

Source

Frameworks history

From academia to industry:

- Theano (U Montreal) -> Tensorflow (Google) ->(?) JAX (Google Deepmind)
- Torch (NYU/Facebook) -> PyTorch (Facebook)
- Caffe (UC Berkeley) -> Caffe2 (Facebook)

Pytorch

What is PyTorch?

A replacement for NumPy:

- which can run on GPUs
- has built-in gradients’ computations

On top of that contains tools and ready-to-use models that make it a flexible research platform.

Side note: pytorch started as an internship project by Adam Paszke.

Tensor operations - as in numpy

```
x = torch.ones(2, 3)
y = torch.rand(2, 3)
print(x + y)
```

```
tensor([[1.6038, 1.8379, 1.3090],
        [1.7103, 1.7608, 1.3079]])
```

Simple placing on GPU

```python
x = torch.ones(2, 3)
x = x.cuda()
print(x)

x = torch.ones(2, 3, device="cuda")
print(x)
```

```
tensor([[1., 1., 1.],
        [1., 1., 1.]], device='cuda:0')
tensor([[1., 1., 1.],
        [1., 1., 1.]], device='cuda:0')
```

Fine-tuning pre-trained models

```python
model = torchvision.models.resnet18(pretrained=True)
print(model.parameters)
```

```
<bound method Module.parameters of ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    ...
  )
  ...
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=512, out_features=1000, bias=True)
)>
```

Based on pytorch [autograd tutorial]

Fine-tuning pre-trained models

```
model = torchvision.models.resnet18(pretrained=True)
for param in model.parameters():
    param.requires_grad = False
# Replace the last fully-connected layer
# Parameters of newly constructed modules have requires_grad=True by default
model.fc = nn.Linear(512, 100)

# Optimize only the classifier
optimizer = optim.SGD(model.fc.parameters(), lr=1e-2, momentum=0.9)
```

Based on pytorch [autograd tutorial]

Gradients

```
import torch

# Create tensors
x = torch.tensor(2.)
w = torch.tensor(3., requires_grad=True)
b = torch.tensor(4., requires_grad=True)

# Define the model
y = w * x + b

# Compute gradients
y.backward()
print(f'dy/dw={w.grad} dy/db={b.grad}')


# Gradient for the next sample
x = torch.tensor(10)
y = w * x + b

w.grad.zero_()
b.grad.zero_()
y.backward()
print(f'dy/dw={w.grad} dy/db={b.grad}')
```

```
dy/dw=2.0 dy/db=1.0
dy/dw=10.0 dy/db=1.0
```

Dataflow Graphs

motivation

Dataflow graphs - why complicate things?

Two main reasons:

- Deploying to embedded devices:
  - Python might not be an option
  - Goal: separate model related code (for inference) from the training code.

- Optimizing performance.

Optimizing performance

- A, B - matrices, v-vector
- When computing AB^T one can transpose B on the fly
- A(Bv) is faster than (AB)v
- Order of matrix multiplications depends on their sizes:
  - Order potentially depends on batch size.

Optimizing performance

Fusion - performing multiple operations at once.

- Matmul + biasadd + relu
  - Perform computationally cheap operation (add bias, relu) when data is still cached.

- Sometimes not all the intermediate results are needed tor the backwards pass.
  - Examples involve convolutions and batchnorm.

Optimizing performance

To enjoy biggest benefits of optimized computation we need to:

- Define upfront the computations to be performed.
- Specify dimensions (like batch size).

Tensorflow 1

Examples in Tensorflow 1.15

Separate graph construction and execution

```
a = tf.constant(3.0, dtype=tf.float32)
b = tf.constant(4.0) # also tf.float32 implicitly
total = a + b
print(a)
print(b)
print(total)
```

Separate graph construction and execution

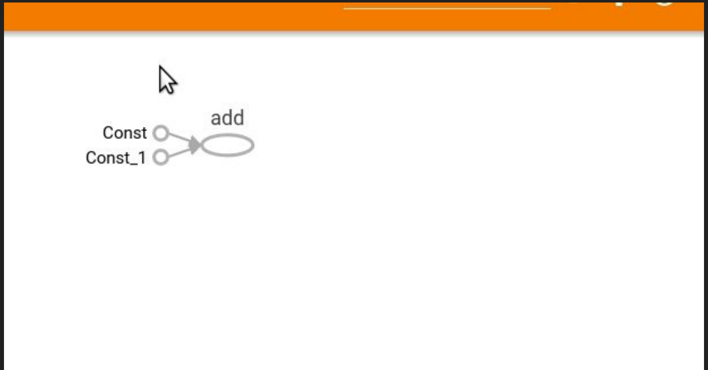
```
a = tf.constant(3.0, dtype=tf.float32)
b = tf.constant(4.0) # also tf.float32 implicitly
total = a + b
print(a)
print(b)
print(total)
```

```
Tensor("Const:0", shape=(), dtype=float32)
Tensor("Const_1:0", shape=(), dtype=float32)
Tensor("add:0", shape=(), dtype=float32)
```

Separate graph construction and execution

```
a = tf.constant(3.0, dtype=tf.float32)
b = tf.constant(4.0) # also tf.float32 implicitly
total = a + b
print(a)
print(b)
print(total)
```

```
Tensor("Const:0", shape=(), dtype=float32)
Tensor("Const_1:0", shape=(), dtype=float32)
Tensor("add:0", shape=(), dtype=float32)
```



Colab link

Separate graph construction and execution

```
a = tf.constant(3.0, dtype=tf.float32)
b = tf.constant(4.0) # also tf.float32 implicitly
total = a + b
print(a)
print(b)
print(total)
```

```
Tensor("Const:0", shape=(), dtype=float32)
Tensor("Const_1:0", shape=(), dtype=float32)
Tensor("add:0", shape=(), dtype=float32)
```

```
sess = tf.Session()
print(sess.run(total))
```

```
7.0
```

Variables

```
my_int_variable = tf.get_variable("my_int_variable", shape=[1, 2, 3],
                                  dtype=tf.int32,
                                  initializer=tf.zeros_initializer)


my_non_trainable = tf.get_variable("my_non_trainable",shape=(),trainable=False)


v = tf.get_variable("v", shape=())
w = tf.get_variable("w", shape=())
x = v + w  # x is a tf.Tensor which is computed based on the values of v and w
```

Variables placement

```
with tf.device("/device:GPU:1"):
  v = tf.get_variable("v", [1])
```

```
cluster_spec = {
    "ps": ["ps0:2222", "ps1:2222"],
    "worker": ["worker0:2222", "worker1:2222", "worker2:2222"]}
with tf.device(tf.train.replica_device_setter(cluster=cluster_spec)):
  v = tf.get_variable("v", shape=[20, 20])  # this variable is placed
                                            # in the parameter server
                                            # by the replica_device_setter
```

Feeding data

A placeholder is a promise to provide a value later, like a function argument.

```
x = tf.placeholder(tf.float32)
y = tf.placeholder(tf.float32)
z = x + y
print(sess.run(z, feed_dict={x: 3, y: 4.5}))
print(sess.run(z, feed_dict={x: [1, 3], y: [2, 4]}))
```

output:

```
7.5
[ 3.  7.]
```

```python
from tensorflow.examples.tutorials.mnist import input_data
import tensorflow as tf

mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

# placeholders for training data
x = tf.placeholder(tf.float32, [None, 784])
y_ = tf.placeholder(tf.float32, [None, 10])

# define the model
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
y = tf.nn.softmax(tf.matmul(x, W) + b)


cross_entropy = -tf.reduce_sum(y_*tf.log(y))
train_step = tf.train.GradientDescentOptimizer(0.01).minimize(cross_entropy)

correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

init = tf.initialize_all_variables()

with tf.Session() as sess:
    sess.run(init)

    for i in range(1000):
        batch_xs, batch_ys = mnist.train.next_batch(100)
        sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})

        if i % 10 == 0:
            print("Model test accuracy")
            print(sess.run(accuracy, feed_dict={x: mnist.test.images, y_: mnist.test.labels}))
```

Based on a tensorflow tutorial

```
from tensorflow.examples.tutorials.mnist import input_data
import tensorflow as tf

mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

# placeholders for training data
x = tf.placeholder(tf.float32, [None, 784])
y_ = tf.placeholder(tf.float32, [None, 10])

# define the model
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
y = tf.nn.softmax(tf.matmul(x, W) + b)


cross_entropy = -tf.reduce_sum(y_*tf.log(y))
train_step = tf.train.GradientDescentOptimizer(0.01).minimize(cross_entropy)

correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

init = tf.initialize_all_variables()

with tf.Session() as sess:
    sess.run(init)

    for i in range(1000):
        batch_xs, batch_ys = mnist.train.next_batch(100)
        sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})

        if i % 10 == 0:
            print("Model test accuracy")
            print(sess.run(accuracy, feed_dict={x: mnist.test.images, y_: mnist.test.labels}))
```

Based on a [tensorflow tutorial]

```
from tensorflow.examples.tutorials.mnist import input_data
import tensorflow as tf

mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

# placeholders for training data
x = tf.placeholder(tf.float32, [None, 784])
y_ = tf.placeholder(tf.float32, [None, 10])

# define the model
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
y = tf.nn.softmax(tf.matmul(x, W) + b)

cross_entropy = -tf.reduce_sum(y_*tf.log(y))
train_step = tf.train.GradientDescentOptimizer(0.01).minimize(cross_entropy)

correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
init = tf.initialize_all_variables()

with tf.Session() as sess:
    sess.run(init)

    for i in range(1000):
        batch_xs, batch_ys = mnist.train.next_batch(100)
        sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})

        if i % 10 == 0:
            print("Model test accuracy")
            print(sess.run(accuracy, feed_dict={x: mnist.test.images, y_: mnist.test.labels}))
```

Based on a tensorflow tutorial

```python
from tensorflow.examples.tutorials.mnist import input_data
import tensorflow as tf

mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

# placeholders for training data
x = tf.placeholder(tf.float32, [None, 784])
y_ = tf.placeholder(tf.float32, [None, 10])

# define the model
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
y = tf.nn.softmax(tf.matmul(x, W) + b)


cross_entropy = -tf.reduce_sum(y_*tf.log(y))
train_step = tf.train.GradientDescentOptimizer(0.01).minimize(cross_entropy)

correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

init = tf.initialize_all_variables()

with tf.Session() as sess:
    sess.run(init)

    for i in range(1000):
        batch_xs, batch_ys = mnist.train.next_batch(100)
        sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})

        if i % 10 == 0:
            print("Model test accuracy")
            print(sess.run(accuracy, feed_dict={x: mnist.test.images, y_: mnist.test.labels}))
```

Based on a [tensorflow tutorial]

```python
from tensorflow.examples.tutorials.mnist import input_data
import tensorflow as tf

mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

# placeholders for training data
x = tf.placeholder(tf.float32, [None, 784])
y_ = tf.placeholder(tf.float32, [None, 10])

# define the model
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
y = tf.nn.softmax(tf.matmul(x, W) + b)


cross_entropy = -tf.reduce_sum(y_*tf.log(y))
train_step = tf.train.GradientDescentOptimizer(0.01).minimize(cross_entropy)

correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

init = tf.initialize_all_variables()

with tf.Session() as sess:
    sess.run(init)

    for i in range(1000):
        batch_xs, batch_ys = mnist.train.next_batch(100)
        sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})

        if i % 10 == 0:
            print("Model test accuracy")
            print(sess.run(accuracy, feed_dict={x: mnist.test.images, y_: mnist.test.labels}))
```

Based on a [tensorflow tutorial]()

```python
from tensorflow.examples.tutorials.mnist import input_data
import tensorflow as tf

mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

# placeholders for training data
x = tf.placeholder(tf.float32, [None, 784])
y_ = tf.placeholder(tf.float32, [None, 10])

# define the model
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
y = tf.nn.softmax(tf.matmul(x, W) + b)


cross_entropy = -tf.reduce_sum(y_*tf.log(y))
train_step = tf.train.GradientDescentOptimizer(0.01).minimize(cross_entropy)

correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

init = tf.initialize_all_variables()

with tf.Session() as sess:
    sess.run(init)

    for i in range(1000):
        batch_xs, batch_ys = mnist.train.next_batch(100)
        sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})

        if i % 10 == 0:
            print("Model test accuracy")
            print(sess.run(accuracy, feed_dict={x: mnist.test.images, y_: mnist.test.labels}))
```

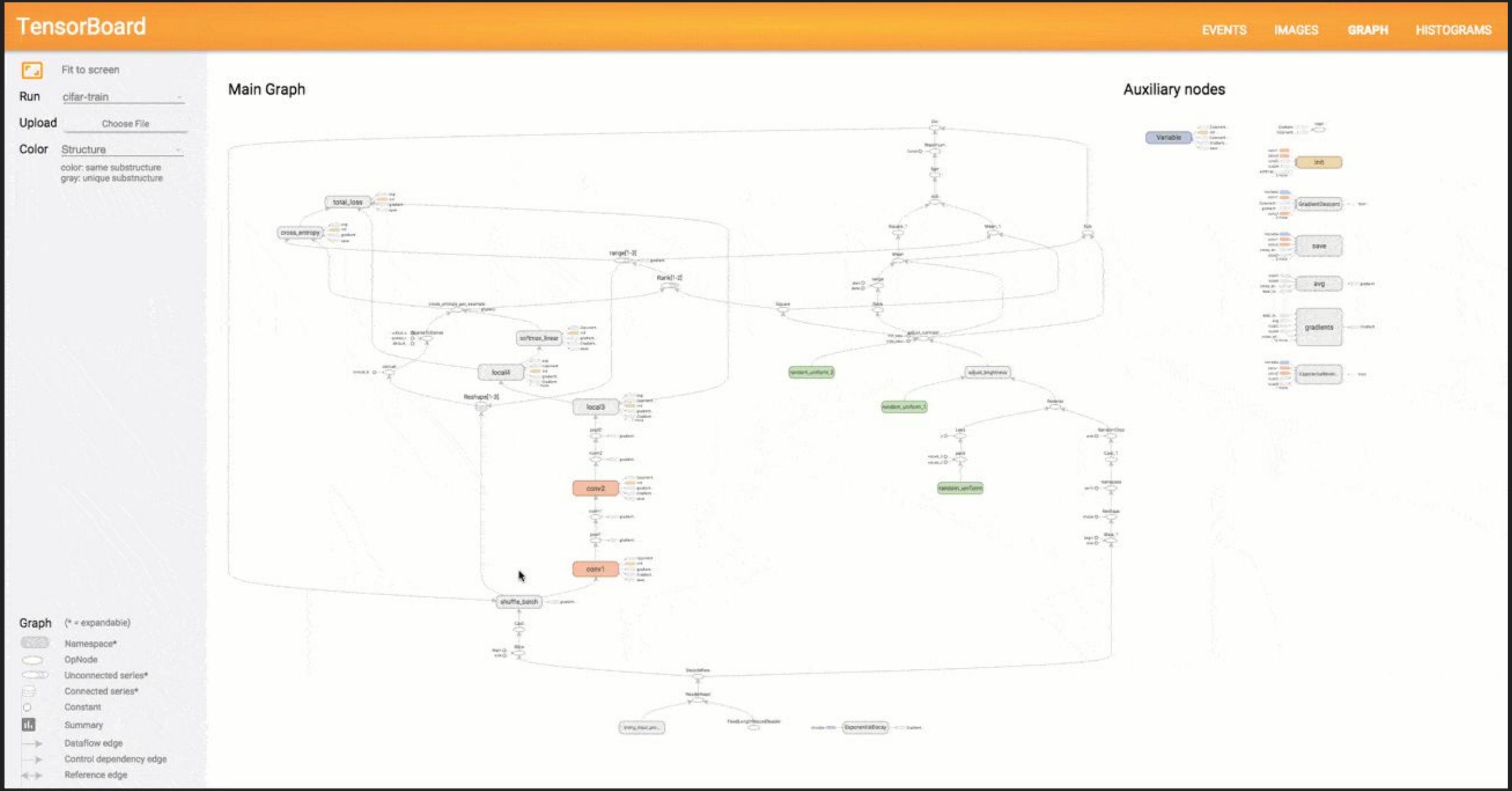
Based on a [tensorflow tutorial]()

Graph visualization

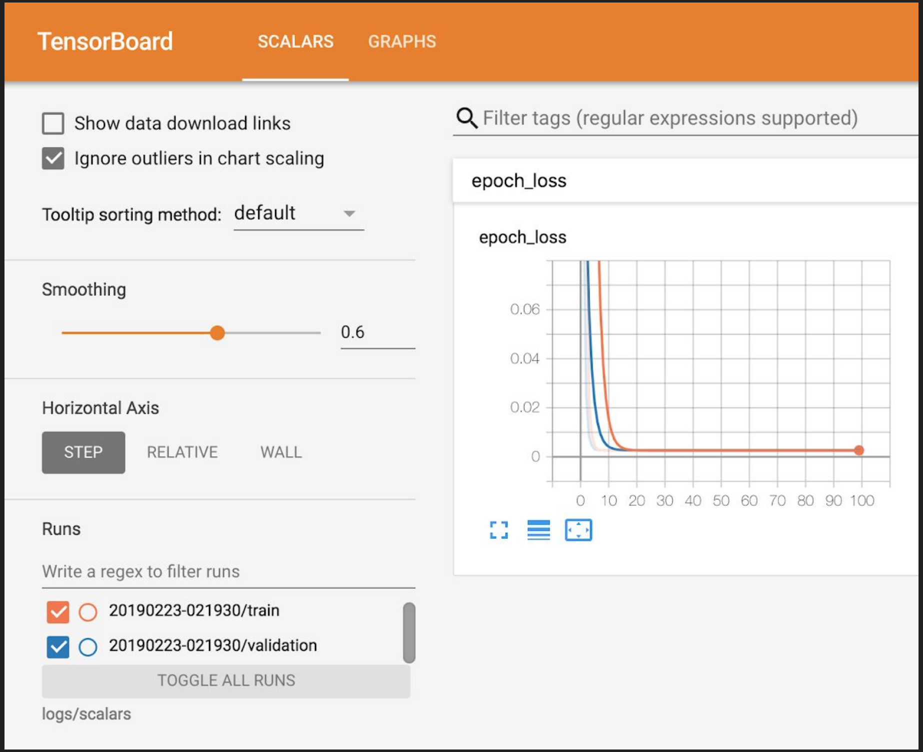
```
writer = tf.summary.FileWriter(logdir='logs')
writer.add_graph(tf.get_default_graph())
writer.flush()
```

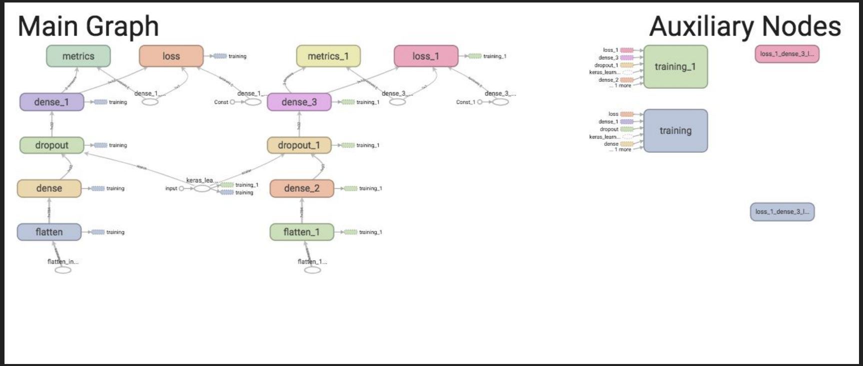
[Colab link]

Graph visualization



Tensorboard





Source

Advantages of dataflow graphs

- Parallelism (independent parts of the graph may be computed in parallel)
- Distributed execution
- Graph optimization (compilation, fused operations)
- Portability and releases (language independent representation)

Disadvantages of Tensorflow 1.X:

- Steep learning curve
- Efficient data loading is non-trivial (queue_runners)
- Mess in higher level API (keras, tflearn, tf.layers, tf-slim, tf.contrib.learn)
- Hard to debug

Tensorflow 2.0 aimed at solving all of those issues
(eager execution, api cleanup, no sessions).

However, for research purposes JAX is more popular than Tensorflow even at Google Brain (and Deepmind).

Static vs Dynamic: Conditional

```python
import torch

def train_mode_computation(x):
    ...

def eval_mode_computation(x):
    ...


if train_mode:
    y = train_mode_computation(x)
else:
    y = eval_mode_computation(x)
```

```python
import tensorflow as tf


def train_mode_computation(x):

    ...


def eval_mode_computation(x):

    ...


train_mode = tf.placeholder(tf.bool, shape=())


y = tf.cond(train_mode, train_mode_computation(x),
            eval_mode_computation(x))
```

Note that `cond` calls `true_fn` and `false_fn` *exactly once* (inside the call to `cond`, and not at all during `Session.run()`). `cond` stitches together the graph fragments created during the `true_fn` and `false_fn` calls with some additional graph nodes to ensure that the right branch gets executed depending on the value of `pred`.

[tf.cond documentation]()

Static vs Dynamic: Loops

Define $y_t = (y_{t-1} + x_t) * w$

```python
n = 10
dim = 5

y0 = torch.randn(dim)
x = torch.randn(n, dim)
w = torch.randn(dim)

y=[y0]
for t in range(n):
  y.append((y[-1] + x[t]) * w)
print(y)
```

```python
import numpy as np

n = 10
dim = 5
x = tf.placeholder(tf.float32, shape=(n, dim))
y0 = tf.placeholder(tf.float32, shape=(dim,))
w = tf.placeholder(tf.float32, shape=(dim,))

def f(prev_y, cur_x):
    return (prev_y + cur_x) * w

y = tf.foldl(f, x, y0)

with tf.Session() as sess:
    values = {
            x: np.random.randn(n, dim),
            y0: np.random.randn(dim),
            w: np.random.randn(dim),
    }
    y_val = sess.run(y, feed_dict=values)
```

Inspired by a slide from CS231n

JAX

JAX:

- seems to be a middle ground between tensorflow and pytorch,
- allows for optimization and training on dedicated hardware (TPUs),
- but is not too far from regular pytorch approach.

JAX like functional programming

- arrays are immutable,

- functions are not allowed to have side-effects
  - consequently randomness requires a special treatment,

- functions are compiled based on anticipated shapes of tensors
  - resizing to result-dependent shape is problematic.

JAX - recommended materials:

- UvA DL notebooks
- JAX tutorials
- JAX for the impatient

How tensors are represented in memory

Why some operations on tensors can be computed in constant time?

How tensors are represented?

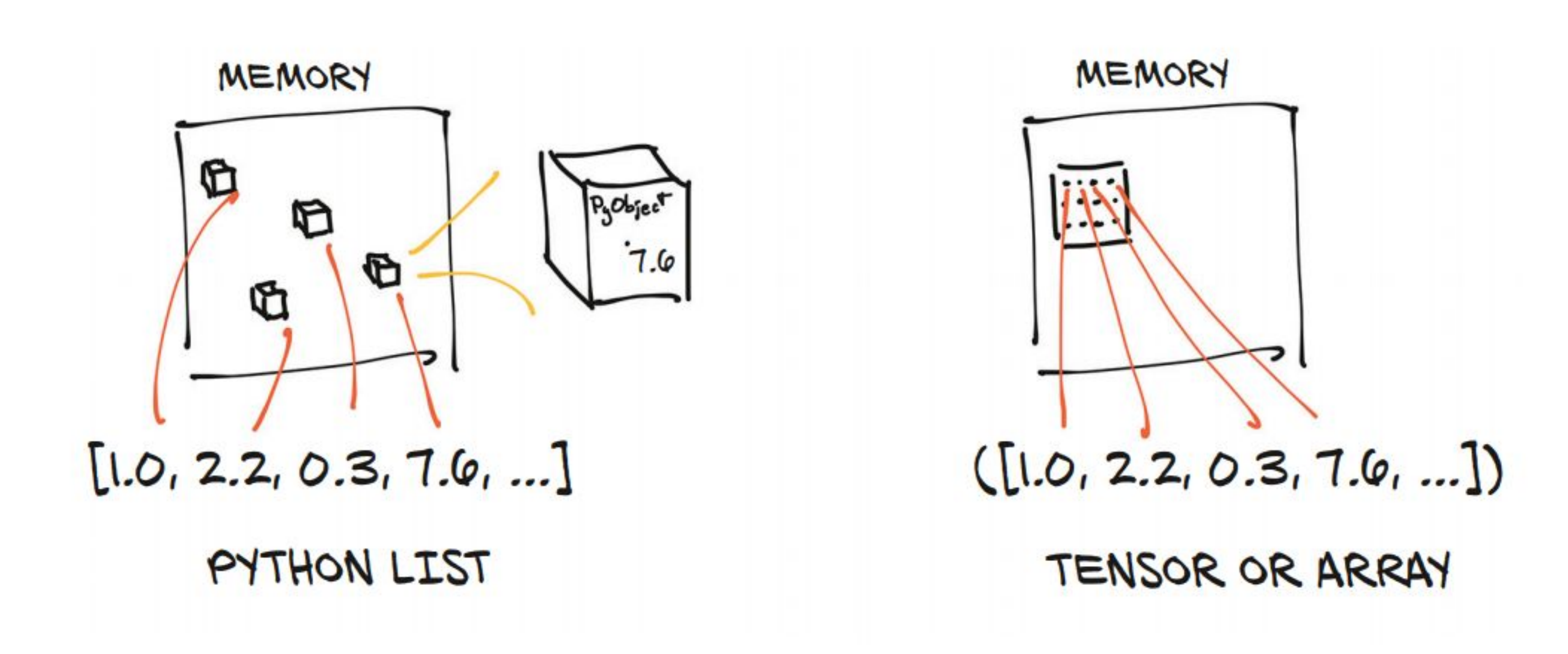


Figure from the [Deep Learning in Pytorch](#) book.

How tensors are represented?

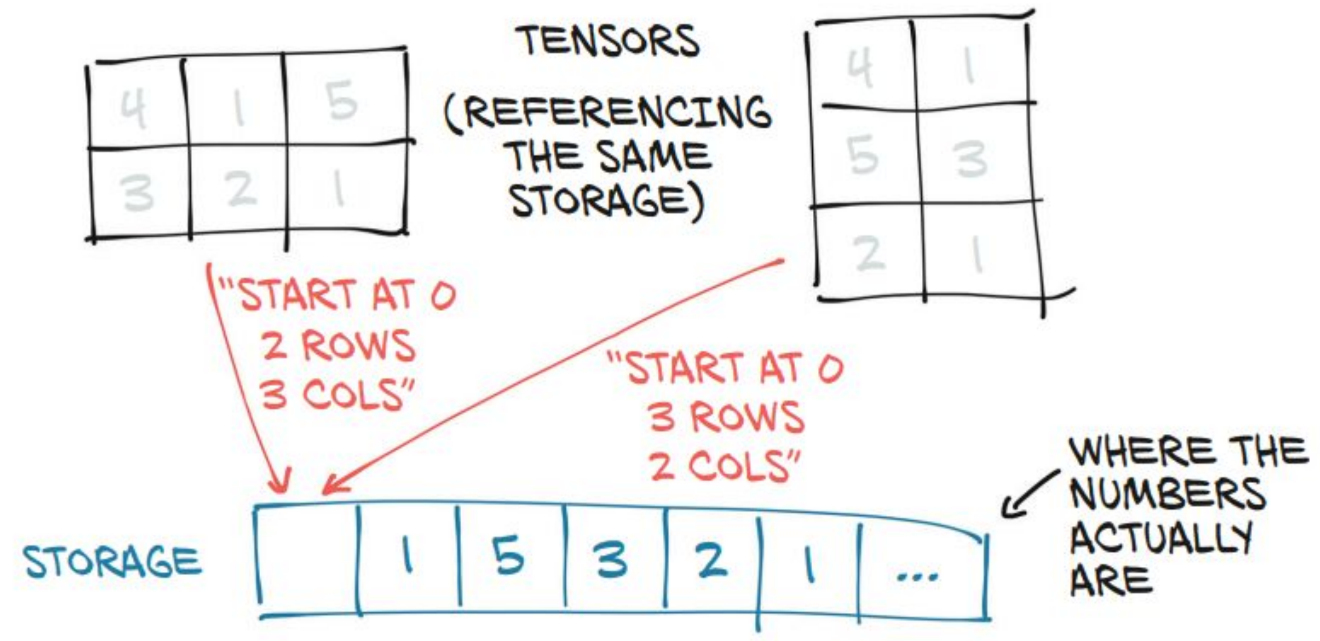


Figure 3.4 Tensors are views of a `Storage` instance.

Figure from the Deep Learning in Pytorch book.

How tensors are represented?

```python
orig = torch.tensor([6, 5, 7, 4, 1, 3, 2, 7, 3, 8])

points = orig[1:].reshape(3,3)

print(f'storage = {list(points.storage())}')
print(f'stride = {points.stride()}')
print(f'storage_offset = {points.storage_offset()}')
```

```
storage = [6, 5, 7, 4, 1, 3, 2, 7, 3, 8]
stride = (3, 1)
storage_offset = 1
```

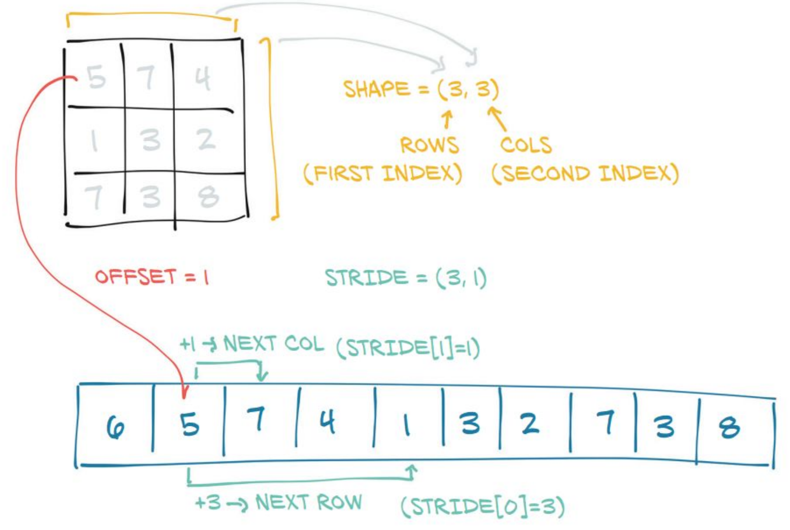


Figure 3.5 Relationship between a tensor's offset, size, and stride. Here the tensor is a view of a larger storage, like one that might have been allocated when creating a larger tensor.

Figure from the [Deep Learning in Pytorch]() book.

How tensors are represented?

```
orig = torch.tensor([6, 5, 3, 1, 2, 4, 1, 7])

points = orig[2:].reshape(2,3).T  # Transposed

print(f'shape = {points.shape}')
print(f'storage_offset = {points.storage_offset()}')
print(f'stride = {points.stride()}')
```

```
shape = torch.Size([3, 2])
storage_offset = 2
stride = (1, 3)
```

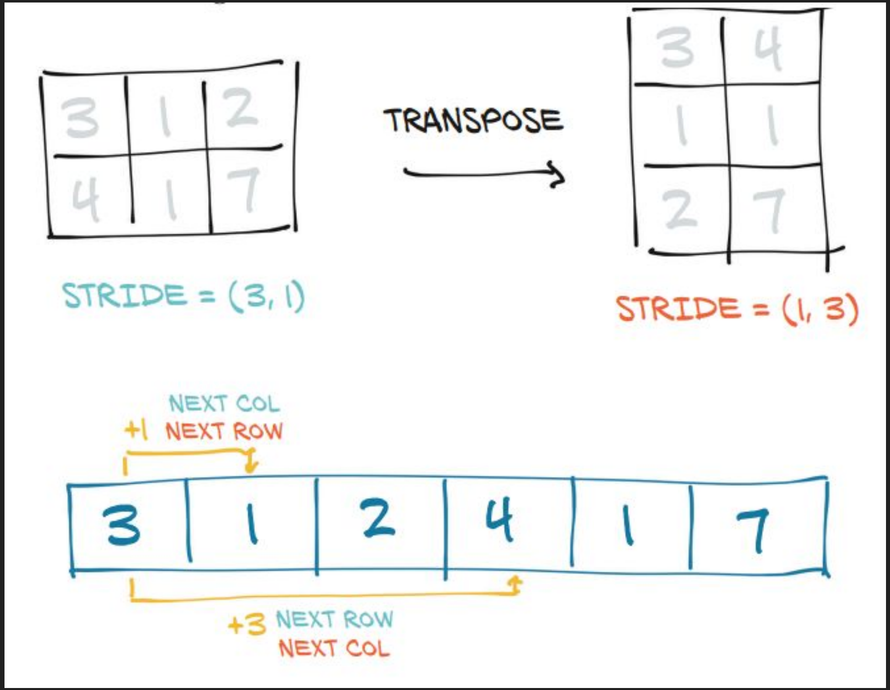


Figure from the [Deep Learning in Pytorch](#) book.

More on PyTorch
during the lab session

https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html

Feedback is a gift

https://tinyurl.com/gsn-2024-11-06