Introduction to PyTorch

TEAMLAB director
최성철

ⓒ NAVER Connect
Foundation

딥러닝을 할 때
코드는 처음부터 다 짠다?

ⓒ NAVER Connect
Foundation

죽을 수도 있습니다.

ⓒ NAVER Connect Foundation

특별기획

뷰노 "의료 AI, 딥러닝 엔진 '뷰노넷' 개발"

[창간 7주년 특별기획 '국내외 지능정보산업'] ⑤업체탐방-뷰노

박시현 기자 | pcsw@bikorea.net

승인 2017.09.21 19:00:10

뷰노는 2014년 창업한 의료 AI 분야 전문업체이다. 삼성종합기술원, LG 등 대기업 연구소에서 다년간 연구원으로 근무했던 시니어급 연구자 6명이 주축이 되어 연구를 이끌어 가고 있다. 현재 전체 인력은 32명이며, 약 70%가 석박사급으로 대부분 기계학습 및 AI 전공자들이다.

◆기업연구소에서 출발, 의료진 보조 AI 개발 역점 = 보통 AI 기술은 기존에 충분히 성숙한 시장에 차별화 포인트로 적용되거나, 부분적인 기술적 향상을 위해 사용되는 경우가 많지만 뷰노는 기존에 불가능하던 제품을 실현하는 기술로 AI를 활용하고 있다.

특히 의료분야는 많은 지식과 전문성이 필요할 뿐만 아니라 사람의 생명을 다루는 분야로서 규제의 벽도 높은 편인데 뷰노는 이런 어려움들을 하나하나 극복해가면서 AI의 새로운 역할을 찾고 있다는 점에서 좋은 평가를 받고 있다.

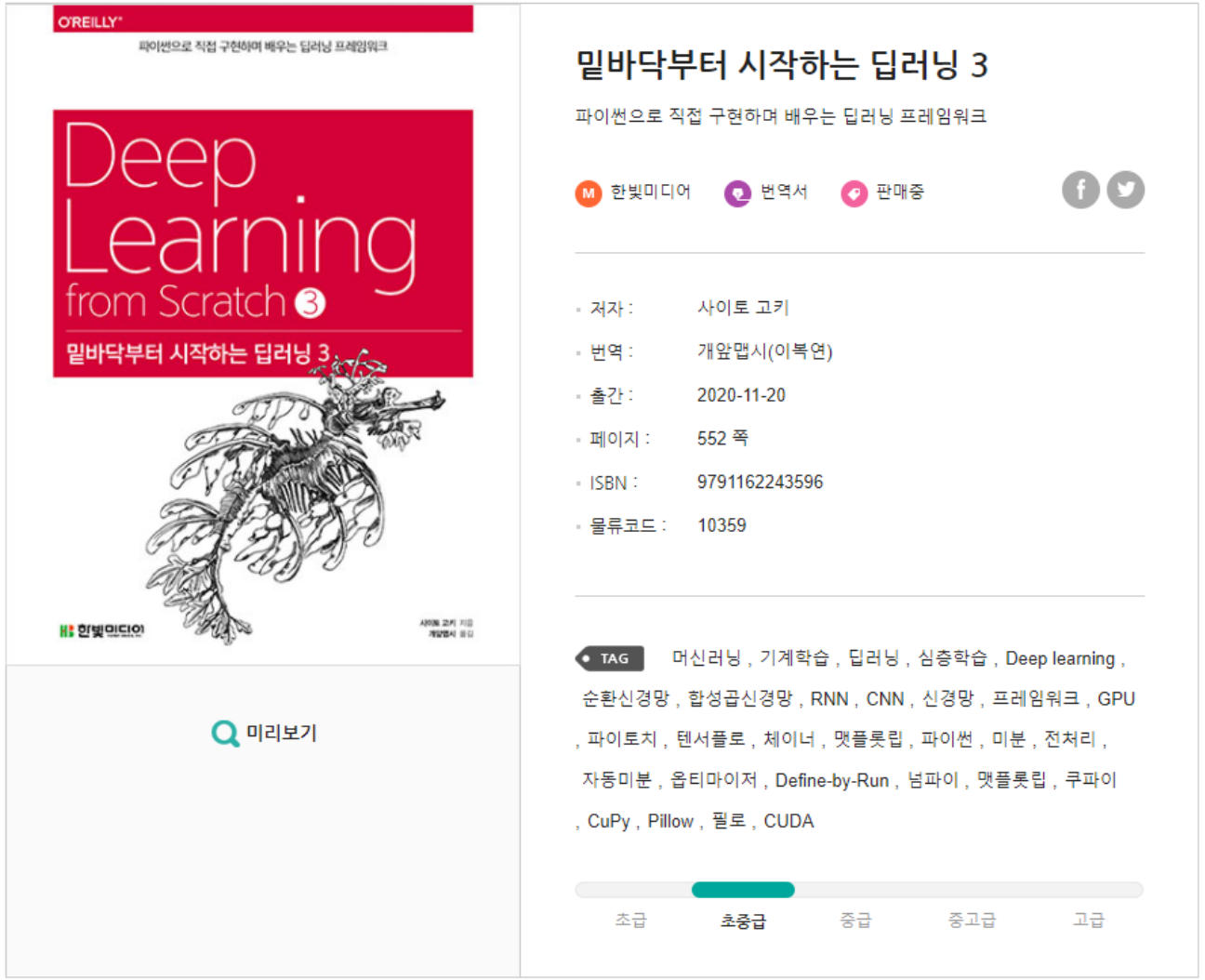
한동대, 오픈소스 딥러닝 프레임워크 공개

대학경제 | 문수빈 기자

2018.07.12 18:09

가 의견 남기기

ⓒ NAVER Connect Foundation



https://www.hanbit.co.kr/store/books/look.php?p_code=B6627606922

ⓒ NAVER Connect Foundation

지금은 남이 만든 걸 씁니다

자료도 많고… 관리도 잘되고… 표준이라서…

ⓒ NAVER Connect Foundation

생각보다 엄청나게 많습니다…

리더는 단 두개!

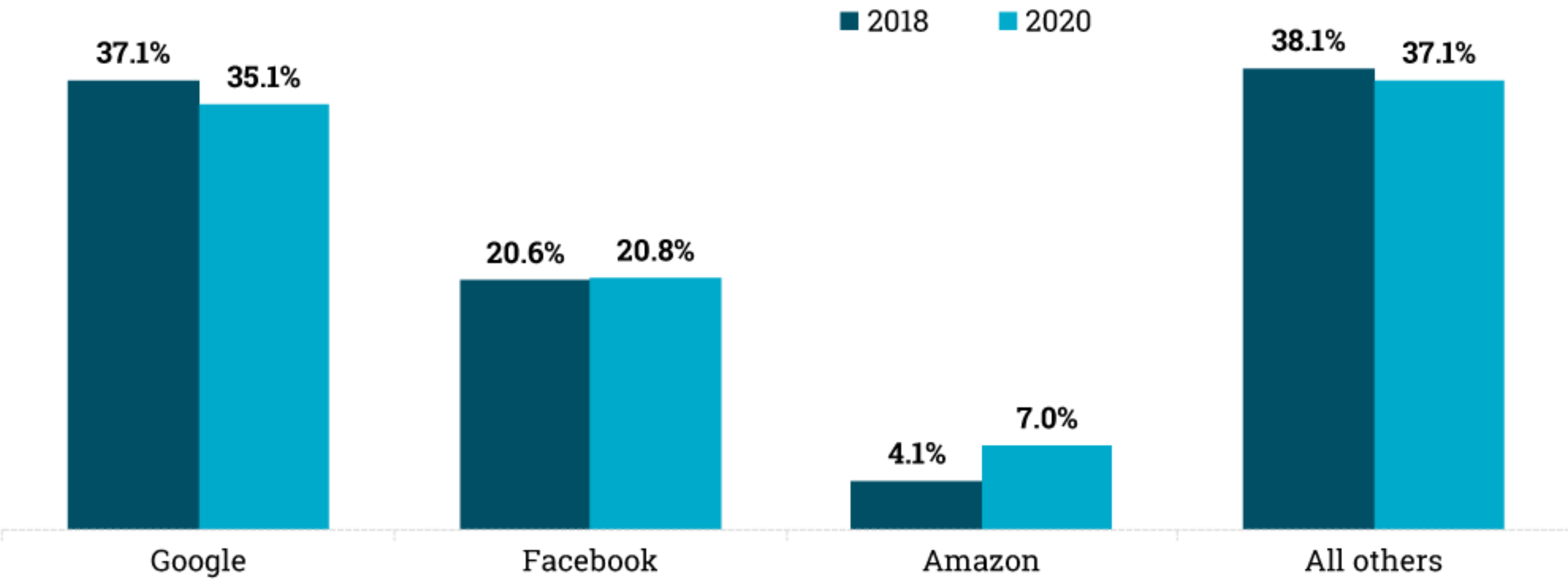
ⓒ NAVER Connect Foundation

리더는 단 두개!

ⓒ NAVER Connect Foundation

Share of US Digital Ad Revenues: Duopoly & Amazon

2018 vs. 2020



Published on MarketingCharts.com in September 2018 | Data Source: eMarketer

Based on estimates from eMarketer, which include all forms of advertising on all internet-connected devices.

	Keras	TensorFlow	PyTorch
Level of API	high-level API[1]	Both high & low level APIs	Lower-level API[2]
Speed	Slow	High	High
Architecture	Simple, more readable and concise	Not very easy to use	Complex[3]
Debugging	No need to debug	Difficult to debugging	Good debugging capabilities
Dataset Compatibility	Slow & Small	Fast speed & large	Fast speed & large datasets
Popularity Rank	1	2	3
Uniqueness	Multiple back-end support	Object Detection Functionality	Flexibility & Short Training Duration
Created By	Not a library on its own	Created by Google	Created by Facebook[4]
Ease of use	User-friendly	Incomprehensive API	Integrated with Python language
Computational graphs used	Static graphs	Static graphs	Dynamic computation graphs[5]

https://datasciencecareer.wordpress.com/2020/12/09/ml03-pytorch-vs-tensorflow/

© NAVER Connect Foundation

Dynamic
Computation Graph

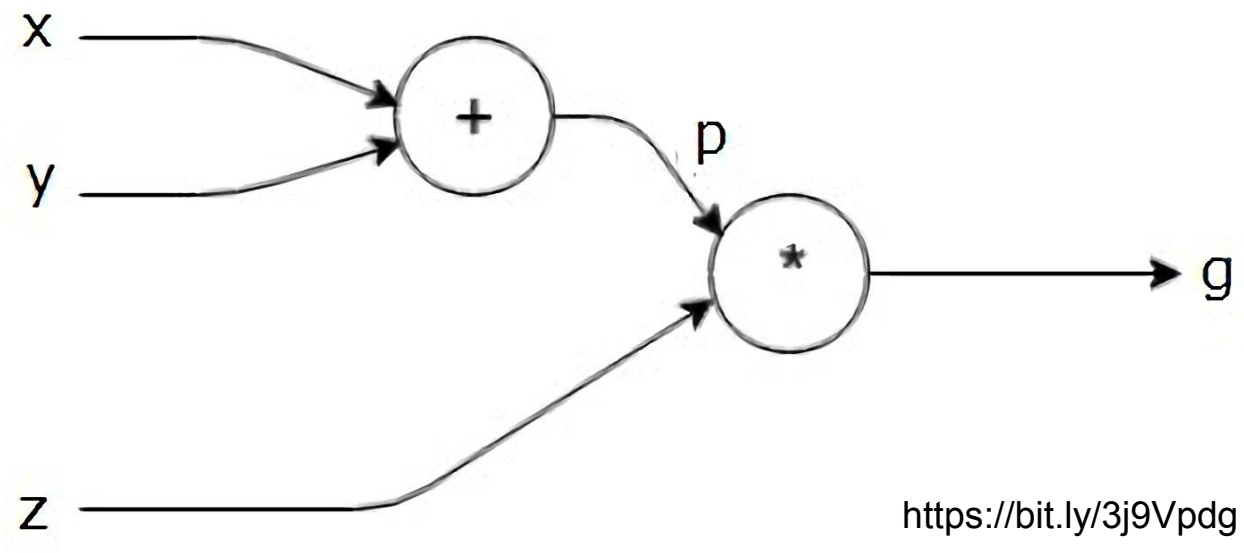
Define and run

© NAVER Connect
Foundation

Computational Graph

- 연산의 과정을 그래프로 표현

$$g = (x + y) * z$$



https://bit.ly/3j9Vpdg

© NAVER Connect Foundation

Computational Graph

- Define and Run
  그래프를 먼저 정의 → 실행시점에 데이터 feed

- Define by Run (Dynamic Computational Graph, DCG)
  실행을 하면서 그래프를 생성하는 방식

ⓒ NAVER Connect Foundation

```
# Start training
with tf.Session() as sess:

    # Run the initializer
    sess.run(init)

    # Fit all training data
    for epoch in range(training_epochs):
        for (x, y) in zip(train_X, train_Y):
            sess.run(optimizer, feed_dict={X: x, Y: y})

        # Display logs per epoch step
        if (epoch+1) % display_step == 0:
            c = sess.run(cost, feed_dict={X: train_X, Y:train_Y})
            print("Epoch:", '%04d' % (epoch+1), "cost=", "{:.9f}".format(c), \
                "W=", sess.run(W), "b=", sess.run(b))

    print("Optimization Finished!")
    training_cost = sess.run(cost, feed_dict={X: train_X, Y: train_Y})
    print("Training cost=", training_cost, "W=", sess.run(W), "b=", sess.run(b), '\n')

    # Graphic display
    plt.plot(train_X, train_Y, 'ro', label='Original data')
    plt.plot(train_X, sess.run(W) * train_X + sess.run(b), label='Fitted line')
    plt.legend()
    plt.show()
```

```
train_X = numpy.asarray([3.3,4.4,5.5,6.71,6.93,4.168,9.779,6.182,7.59,2.167,
                         7.042,10.791,5.313,7.997,5.654,9.27,3.1])
train_Y = numpy.asarray([1.7,2.76,2.09,3.19,1.694,1.573,3.366,2.596,2.53,1.221,
                         2.827,3.465,1.65,2.904,2.42,2.94,1.3])
n_samples = train_X.shape[0]

# tf Graph Input
X = tf.placeholder("float")
Y = tf.placeholder("float")

# Set model weights
W = tf.Variable(rng.randn(), name="weight")
b = tf.Variable(rng.randn(), name="bias")

# Construct a linear model
pred = tf.add(tf.multiply(X, W), b)

# Mean squared error
cost = tf.reduce_sum(tf.pow(pred-Y, 2))/(2*n_samples)
# Gradient descent
#  Note, minimize() knows to modify W and b because Variable objects are trainable=True by default
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)
```

Foundation

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture8.pdf

Static vs Dynamic Graphs

TensorFlow: Build graph once, then run many times (**static**)

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

learning_rate = 1e-5
new_w1 = w1.assign(w1 - learning_rate * grad_w1)
new_w2 = w2.assign(w2 - learning_rate * grad_w2)
updates = tf.group(new_w1, new_w2)

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    values = {x: np.random.randn(N, D),
              y: np.random.randn(N, D),}
    losses = []
    for t in range(50):
        loss_val, _ = sess.run([loss, updates],
                               feed_dict=values)
```

Build graph

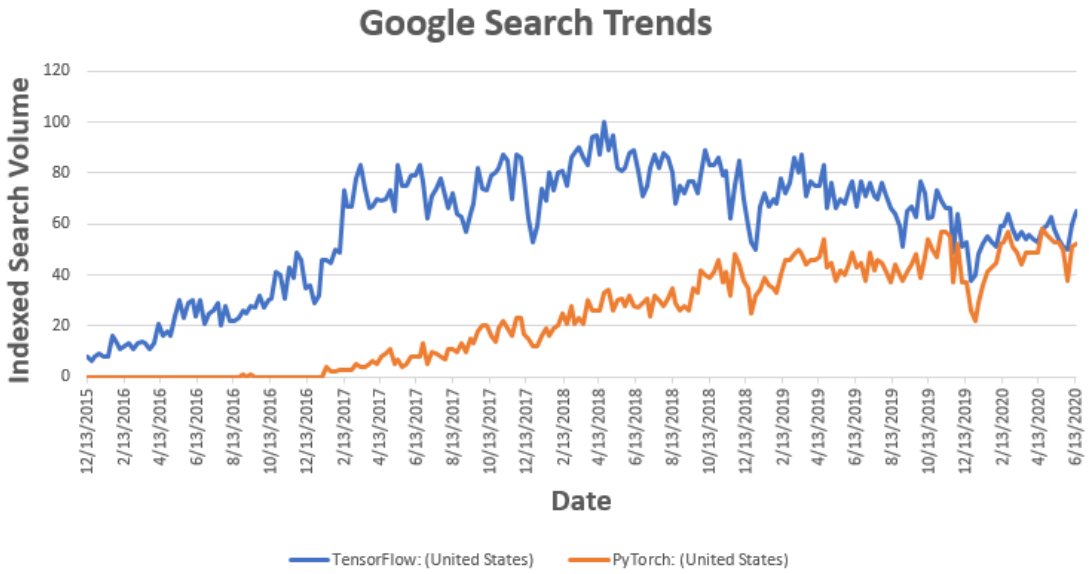
Run each iteration

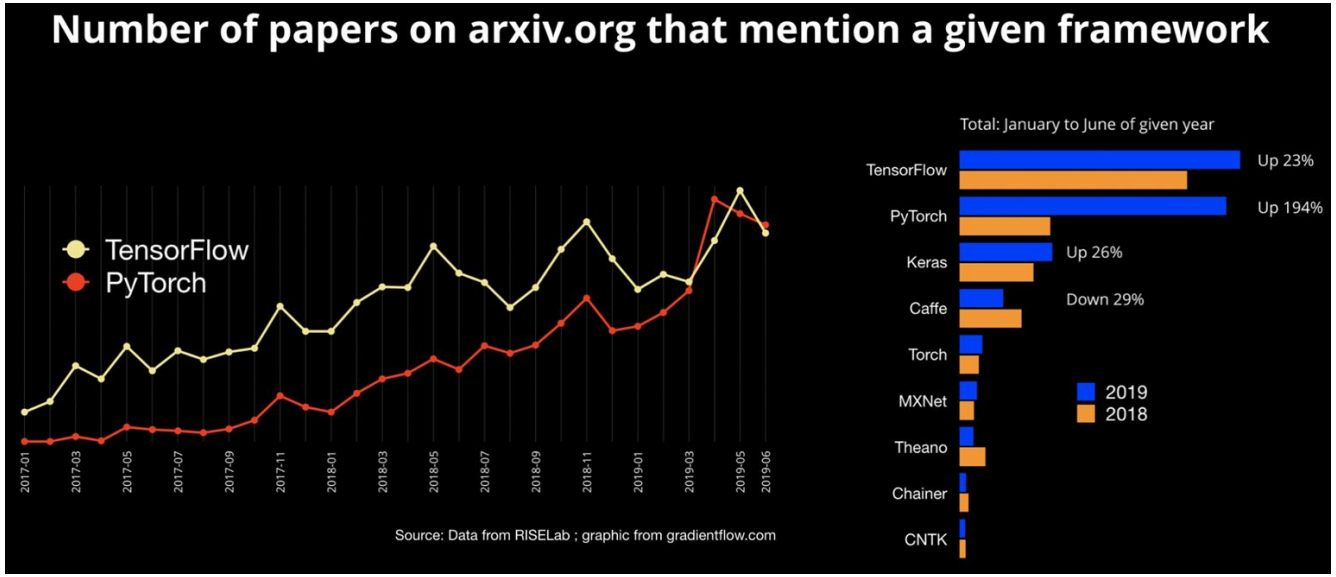
PyTorch: Each forward pass defines a new graph (**dynamic**)

```
import torch
from torch.autograd import Variable

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

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

    if w1.grad: w1.grad.data.zero_()
    if w2.grad: w2.grad.data.zero_()
    loss.backward()

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

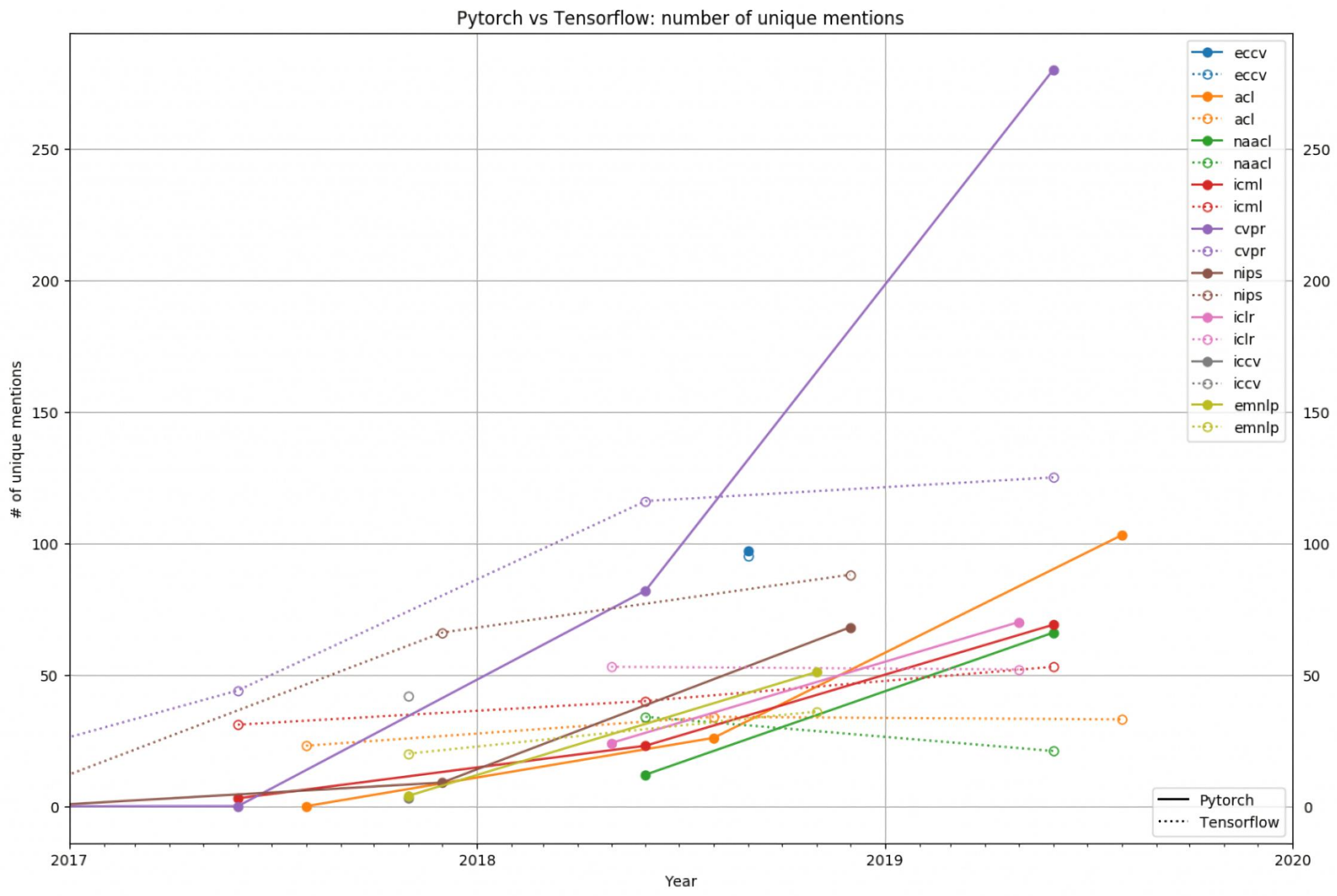
New graph each iteration

© NAVER Connect Foundation





© NAVER Connect Foundation



https://bit.ly/2Vb0VnY

Why PyTorch

- Define by Run 의 장점

  즉시 확인 가능 → pythonic code

- GPU support, Good API and community
- 사용하기 편한 장점이 가장 큼
- TF는 production 과 scalability의 장점

ⓒ NAVER Connect Foundation

Numpy + AutoGrad + Function

© NAVER Connect Foundation

- Numpy 구조를 가지는 Tensor 객체로 array 표현
- 자동미분을 지원하여 DL 연산을 지원
- 다양한 형태의 DL을 지원하는 함수와 모델을 지원함

ⓒ NAVER Connect Foundation

End of Document

Thank You.

ⓒ NAVER Connect Foundation