cs231n 2017
Deep learning software

Lecture 8

Why GPU?

CPU vs GPU

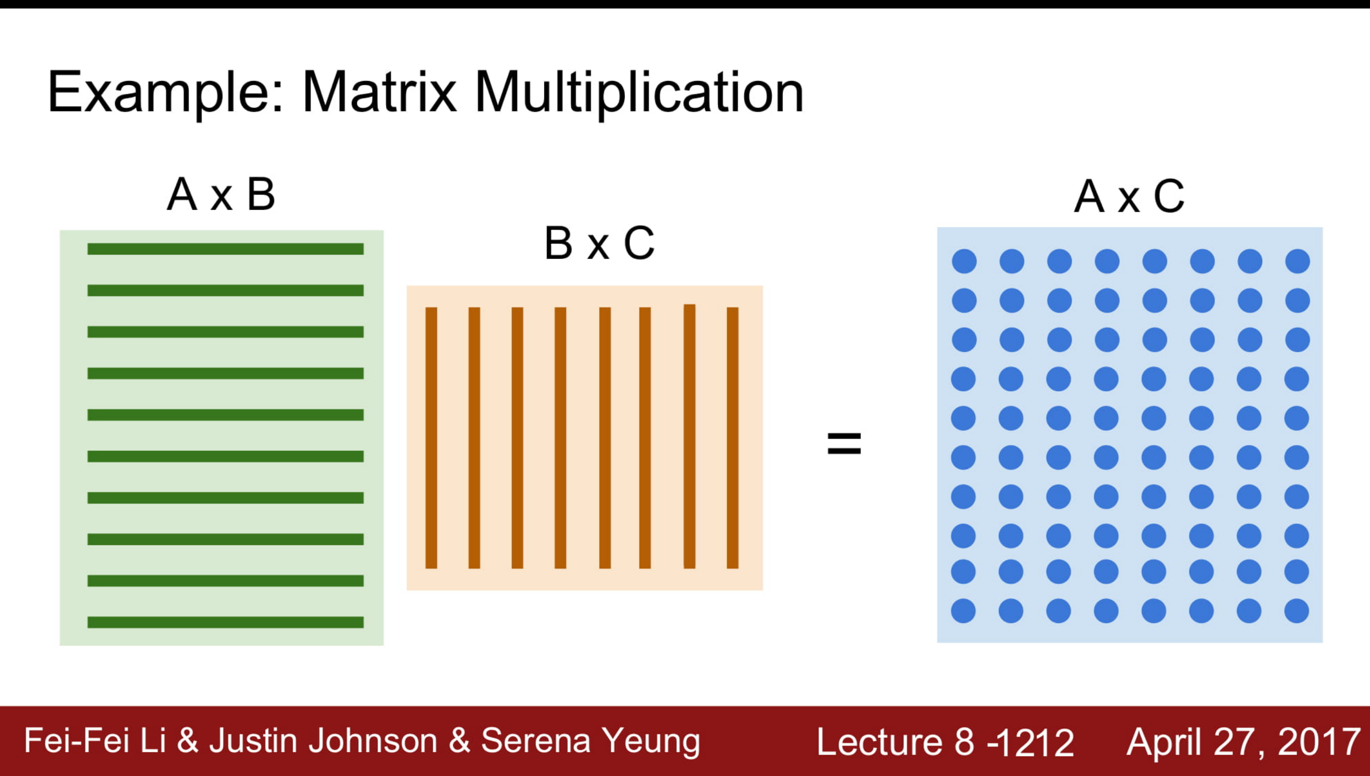
	# Cores	Clock Speed	Memory	Price
CPU (Intel Core i7-7700k)	4 (8 threads with hyperthreading)	4.4 GHz	Shared with system	$339
CPU (Intel Core i7-6950X)	10 (20 threads with hyperthreading)	3.5 GHz	Shared with system	$1723
GPU (NVIDIA Titan Xp)	3840	1.6 GHz	12 GB GDDR5X	$1200
GPU (NVIDIA GTX 1070)	1920	1.68 GHz	8 GB GDDR5	$399

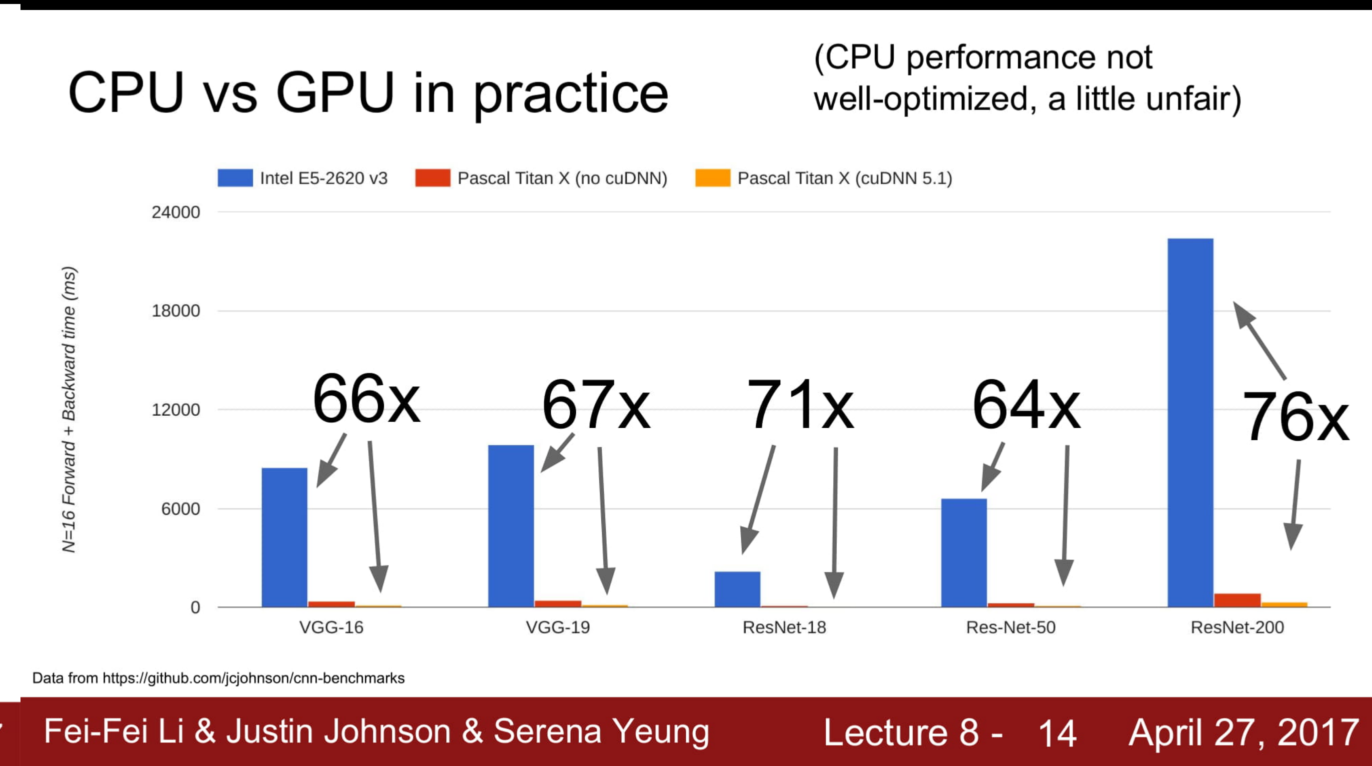
CPU: Fewer cores, but each core is much faster and much more capable; great at sequential tasks

GPU: More cores, but each core is much slower and "dumber"; great for parallel tasks

- GPU는 Core 하나의 성능은 낮지만, 많은양의 core로 병렬 처리를 잘 처리함.

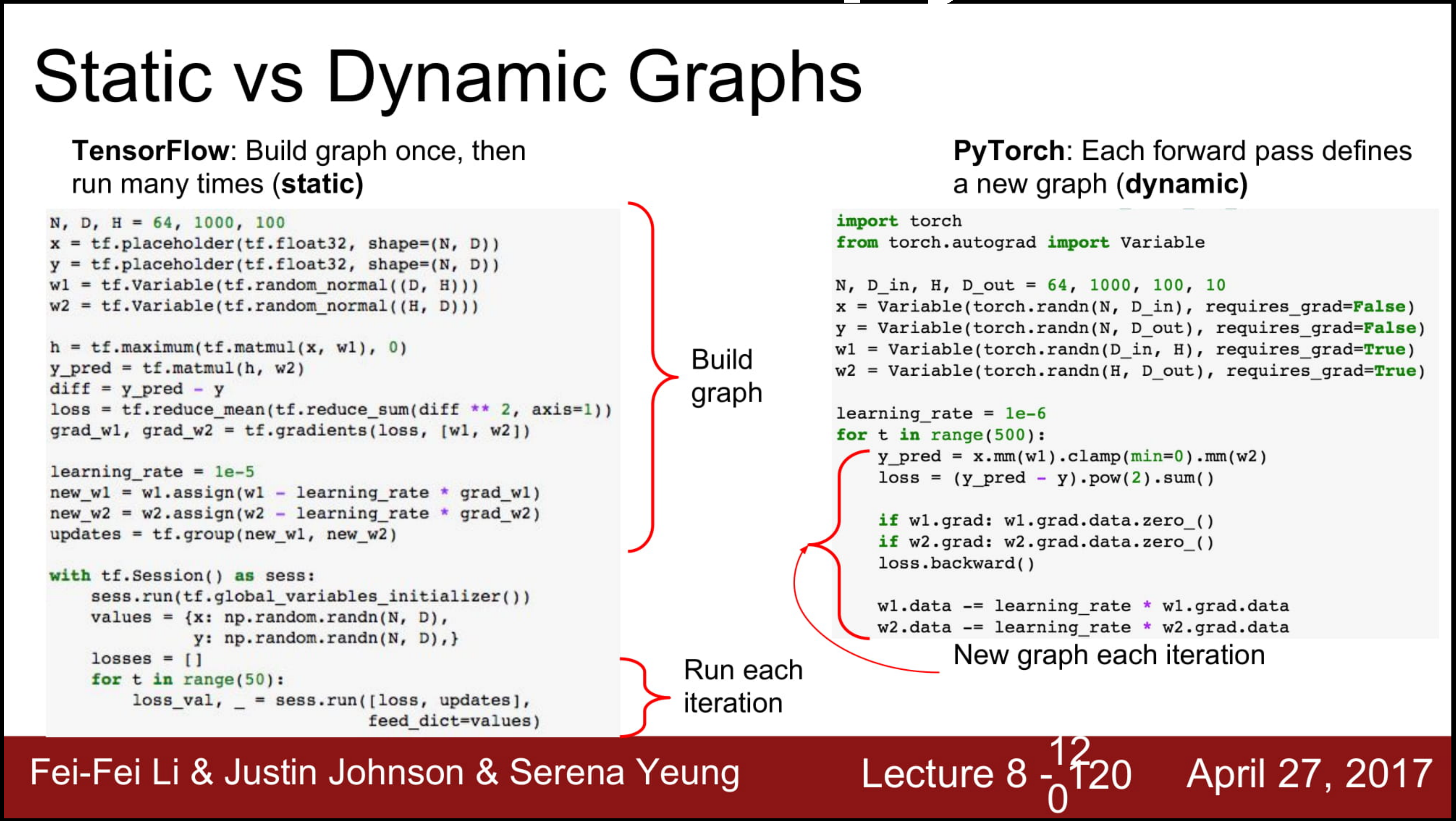
Why GPU?





- Matrix multiplication은 많은 양의 연산을 요구함.
- CPU는 core수가 적어서 병렬 처리를 하지 못함.GPU는 많은 양의 core로 병렬처리를 하기 때문에 Matrix 곱을 빠르게 처리 가능

Tensorflow, pyTorch



- Tensorflow와 pyTorch는 구조적으로 많이 다르다.
- tensorflow는 define & run, pyTorch는 즉시 실행 이라는 다른 그래프 실행 구조를 가진다.

Static, dynamic graph

Static vs Dynamic: Conditional

$$y = \begin{cases} w1 * x & \text{if } z > 0 \\ w2 * x & \text{otherwise} \end{cases}$$

PyTorch: Normal Python

```
N, D, H = 3, 4, 5

x = Variable(torch.randn(N, D))
w1 = Variable(torch.randn(D, H))
w2 = Variable(torch.randn(D, H))

z = 10
if z > 0:
    y = x.mm(w1)
else:
    y = x.mm(w2)
```

TensorFlow: Special TF control flow operator!

```
N, D, H = 3, 4, 5
x = tf.placeholder(tf.float32, shape=(N, D))
z = tf.placeholder(tf.float32, shape=None)
w1 = tf.placeholder(tf.float32, shape=(D, H))
w2 = tf.placeholder(tf.float32, shape=(D, H))

def f1(): return tf.matmul(x, w1)
def f2(): return tf.matmul(x, w2)
y = tf.cond(tf.less(z, 0), f1, f2)

with tf.Session() as sess:
    values = {
        x: np.random.randn(N, D),
        z: 10,
        w1: np.random.randn(D, H),
        w2: np.random.randn(D, H),
    }
    y_val = sess.run(y, feed_dict=values)
```

Static, dynamic graph

- Static graph는 간단한 python if 구문을 graph에 넣을려면, define단계에서 미리 정의해둬야함.
- Dynamic graph는 graph가 실행되는 단계(runtime)에서 if 구문을 통해 graph 구조를 변경할 수 있음.

Static, dynamic graph

Static vs Dynamic: Loops

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

PyTorch: Normal Python

```
T, D = 3, 4
y0 = Variable(torch.randn(D))
x = Variable(torch.randn(T, D))
w = Variable(torch.randn(D))

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

TensorFlow: Special TF control flow

```
T, N, D = 3, 4, 5
x = tf.placeholder(tf.float32, shape=(T, D))
y0 = tf.placeholder(tf.float32, shape=(D,))
w = tf.placeholder(tf.float32, shape=(D,))

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(T, D),
        y0: np.random.randn(D),
        w: np.random.randn(D),
    }
    y_val = sess.run(y, feed_dict=values)
```

Static, dynamic graph

- Static graph는 for 문을 graph에 넣을려면 functional programming 처럼 기능을 함수로 만들고 함수를 넘겨야함.
- Dynamic graph는 python 짜는거 처럼 짜면 됨. 알아서 graph를 만들어줌.

Static, dynamic graph

Static vs Dynamic: Optimization

With static graphs, framework can **optimize** the graph for you before it runs!

The graph you wrote

Conv
ReLU
Conv
ReLU
Conv
ReLU

Equivalent graph with **fused operations**

Conv+ReLU
Conv+ReLU
Conv+ReLU

Static, dynamic graph

- Static graph의 장점은 마치 compile 언어 처럼 tensorflow가 모델을 optimization해줄 수 있다는것이다.
- 또한 한번 model 을 정의하고 model이 변하지 않으므로 training 할때 model을 불러오기만 하면 되서 편하다.