MXNet

a rushed introduction

what's there to talk about?

- what's this thing?
- competition
- imperative vs symbolic
- example networks (gluon)
- fast
- parallel
- resources

what's this thing?

- c++; many bindings (Python, Scala, Spark, ..)
- scales very well (almost linearly)
- Amazon bought team; used internally (services are being migrated)

Competition

Top libraries by Github issues opened

#1:	8370	tensorflow/tensorflow
#2:	5806	fchollet/keras
#3:	4558	dmlc/mxnet
#4:	3908	BVLC/caffe
#5:	2465	Theano/Theano
#6:	2462	baidu/paddle
#7:	2264	deeplearning4j/deeplearning4j
#8:	2124	Microsoft/CNTK
#9:	1601	pytorch/pytorch
#10:	1139	NVIDIA/DIGITS
#11:	1005	pfnet/chainer
#12:	738	caffe2/caffe2
#13:	709	tflearn/tflearn
#14:	664	davisking/dlib
#15:	575	torch/torch7
#16:	488	Lasagne/Lasagne
#17:	469	clab/dynet
#18:	324	NervanaSystems/neon
#19:	47	deepmind/sonnet

Top libraries by Github stars

#1:	71627	tensorflow/tensorflow
#2:	20489	BVLC/caffe
#3:	20038	fchollet/keras
#4:	12558	Microsoft/CNTK
#5:	11369	dmlc/mxnet
#6:	7712	pytorch/pytorch
#7:	7332	torch/torch7
#8:	7297	deeplearning4j/deeplearning4j
#9:	6981	Theano/Theano
#10:	6767	tflearn/tflearn
#11:	5742	caffe2/caffe2
#12:	5544	baidu/paddle
#13:	5336	deepmind/sonnet
#14:	3242	Lasagne/Lasagne
#15:	3232	NervanaSystems/neon
#16:	2987	pfnet/chainer
#17:	2833	davisking/dlib
#18:	2525	NVIDIA/DIGITS
#19:	1775	clab/dynet

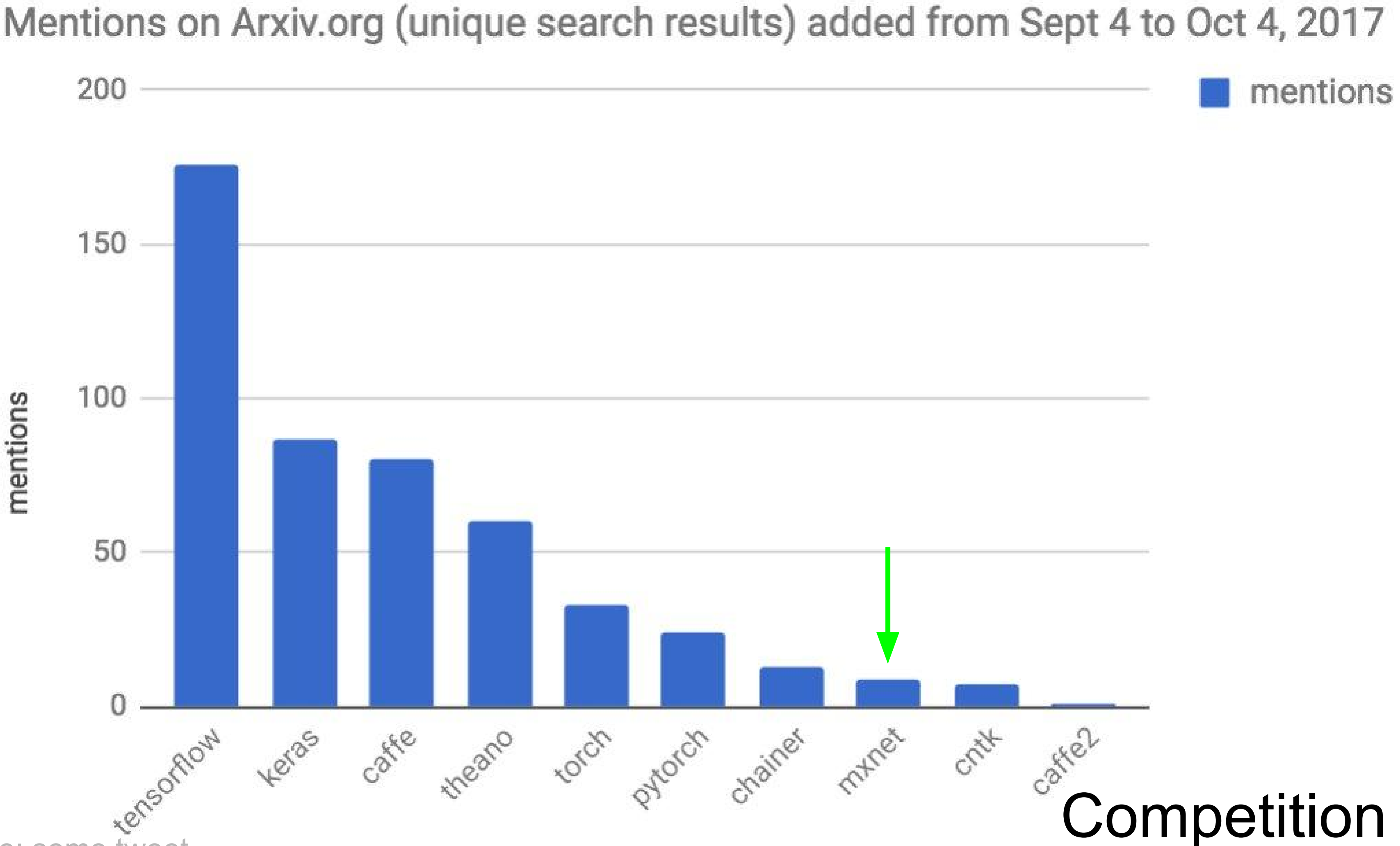
Top libraries by Github contributors

#1:	1079	tensorflow/tensorflow
#2:	537	fchollet/keras
#3:	432	dmlc/mxnet
#4:	322	Theano/Theano
#5:	318	pytorch/pytorch
#6:	249	BVLC/caffe
#7:	149	Microsoft/CNTK
#8:	139	pfnet/chainer
#9:	134	torch/torch7
#10:	125	deeplearning4j/deeplearning4j
#11:	125	caffe2/caffe2
#12:	104	tflearn/tflearn
#13:	84	clab/dynet
#14:	79	davisking/dlib
#15:	77	baidu/paddle
#16:	70	NervanaSystems/neon
#17:	62	Lasagne/Lasagne
#18:	42	NVIDIA/DIGITS
#19:	16	deepmind/sonnet

Top libraries by Github forks

#1:	35371	tensorflow/tensorflow
#2:	12575	BVLC/caffe
#3:	7293	fchollet/keras
#4:	4256	dmlc/mxnet
#5:	3659	deeplearning4j/deeplearning4j
#6:	3272	Microsoft/CNTK
#7:	2302	Theano/Theano
#8:	2166	torch/torch7
#9:	1599	pytorch/pytorch
#10:	1483	baidu/paddle
#11:	1450	tflearn/tflearn
#12:	1278	caffe2/caffe2
#13:	974	davisking/dlib
#14:	921	NVIDIA/DIGITS
#15:	908	Lasagne/Lasagne
#16:	798	pfnet/chainer
#17:	716	NervanaSystems/neon
#18:	656	deepmind/sonnet
#19:	447	clab/dynet

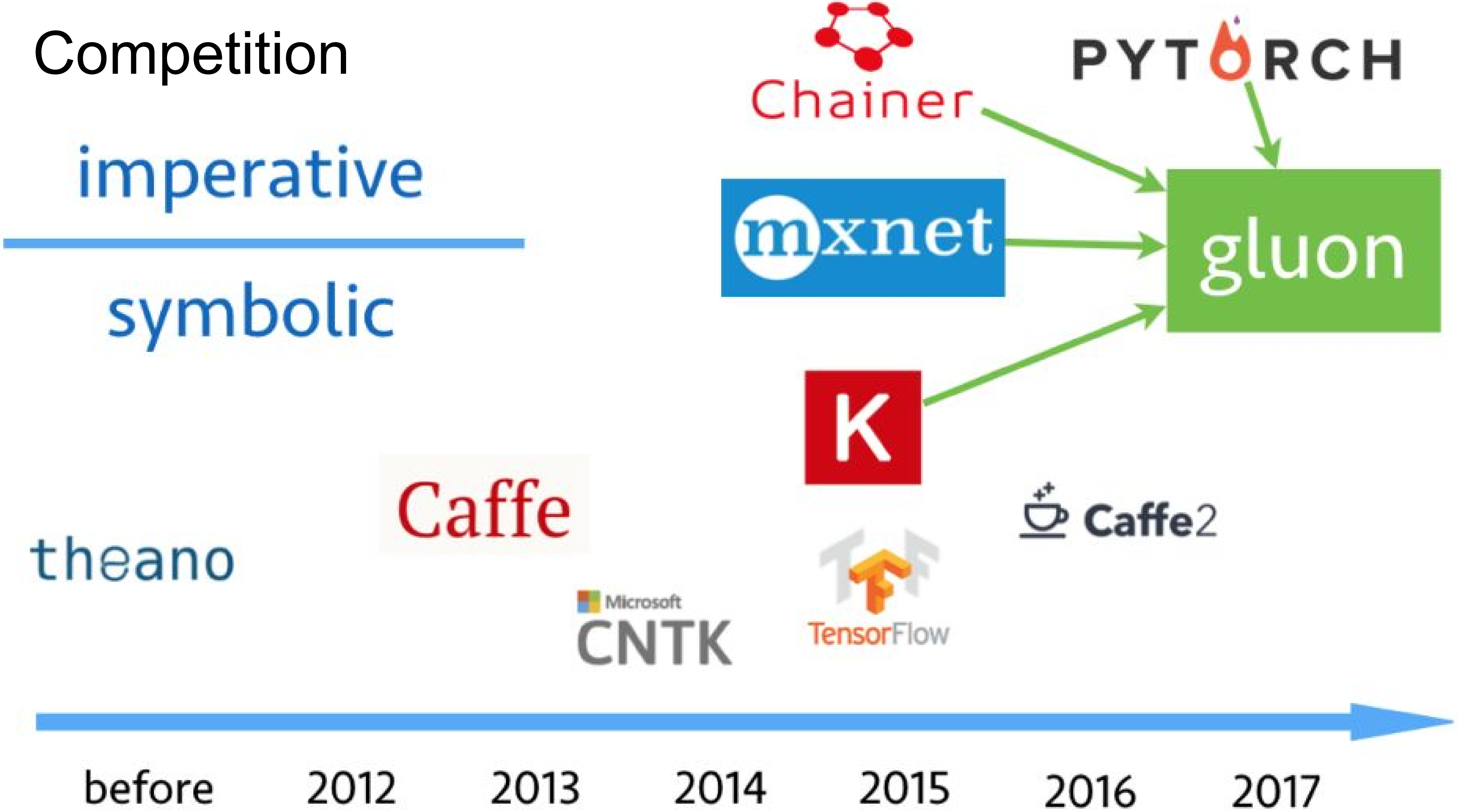
source: some tweet



Competition

source: some tweet

Competition



source: the straight dope (slides)

imperative vs symbolic

NDArray: numpy-like imperative linear algebra

- automatic differentiation:
  only define forward pass, get backward pass for free
- async (non-blocking parallel)

like: PyTorch

```
from mxnet import nd, autograd

s = nd.array([8.]);  x =
nd.arange(4)
X = nd.ones((2, 3))
Y = nd.array([[1,2,3], [4,5,6]])
Z = X * Y     # <- answer!

Y.attach_grad()
with autograd.record():
  A = Y * 2   # e.g., inference
  B = A * Y   # e.g., loss function
B.backward()
print(Y.grad) # <- gradient!
```

imperative vs **symbolic**

sym: numpy-like symbolic linear algebra

- automatic differentiation: only define forward pass, get backward pass for free
- define full graph, then compile, then evaluate with data
- can request json (`net.tojson`), or visualise network (`mxnet.viz.plot_network(net)`)

like: TensorFlow, Caffe, Theano

```
from mxnet import sym, nd, autograd

s = sym.var('s');  x = sym.arange(4)
X = sym.ones((2, 3))
Y = sym.var('Y')
Z = X * Y      # <- symbolic graph!

A = Y * 2      # e.g., inference
B = A * Y      # e.g., loss function

Ydata = nd.array([[1,2,3], [4,5,6]])
Ygrad = nd.empty(Ydata.shape)
exc = B.bind(cpu(),
             args={'Y': Ydata},
             args_grad={'Y': Ygrad})
r = exc.forward() # evaluate graph
exc.backward(
  out_grads=nd.ones(r[0].shape))
print(Ygrad)  # <- gradient!
```

Example: hello world

```
ctx = mx.cpu()
n_in = 2; n_out = 1; n_ex = int(1e4)
X = nd.random_normal(shape=(n_ex, n_in))
noise = .01 * nd.random_normal(shape=(n_ex,))
y = real_fn(X) + noise
```

```
# gluon
net = gluon.nn.Sequential()
with net.name_scope():
  net.add(gluon.nn.Dense(n_out))

net.collect_params().initialize(
  mx.init.Normal(sigma=1.), ctx=ctx)


loss = gluon.loss.L2Loss()


sgb = gluon.Trainer(net.collect_params(),
                    'sgd',
                    {'learning_rate': lr})
```

```
# alt: from scratch (for custom needs)
w = nd.random_normal(shape=(n_in, n_out)
b = nd.random_normal(shape=n_out)
params = [w, b]
w.attach_grad();  b.attach_grad()

def net(X):
  return nd.dot(X, w) + b

def loss(yhat, y):
  return nd.mean((yhat - y) ** 2)

def sgd(params, lr):
  for param in params:
    parma[:] = parma - lr * param.grad
```

Example: hello world

```
ctx = mx.cpu()
n_in = 2; n_out = 1; n_ex = int(1e4)
X = nd.random_normal(shape=(n_ex, n_in))
noise = .01 * nd.random_normal(shape=(n_ex,))
y = real_fn(X) + noise
```

```
# gluon
net = gluon.nn.Sequential()
with net.name_scope():
  net.add(gluon.nn.Dense(n_out))

net.collect_params().initialize(
  mx.init.Normal(sigma=1.), ctx=ctx)


loss = gluon.loss.L2Loss()


sgb = gluon.Trainer(net.collect_params(),
                    'sgd',
                    {'learning_rate': lr})
```

```
# train!
train_d = gluon.data.DataLoader(
  gluon.data.ArrayDataset(X, y),
  batch_size=batch_size, shuffle=True)

for e in range(epochs):
  for i, (data, label) in enumerate(train_d):
    with autograd.record():
      output = net(data)
      _loss = loss(output, label)
    _loss.backward()
    sgd.step(batch_size)

  print('curr loss:',
        nd.mean(_loss).assscalar())
```

Example: CNN

```
ctx = mx.cpu()
n_out = 1
tr = lambda data, label: (nd.transpose(data.astype(np.float32), (2, 0, 1)/255,
                          label.astype(np.float32)
mnist_train = gluon.data.vision.MNIST(train=True, transform=tr)
```

```
# gluon
net = gluon.nn.Sequential()
with net.name_scope():
  net.add(gluon.nn.Conv2D(channels=20,activation='relu'))
  net.add(gluon.nn.MaxPool2D(pool_size=2, strides=2))
  net.add(gluon.nn.Flatten())
  net.add(gluon.nn.Dense(n_out))
  net.add(gluon.nn.Dense(n_out))

net.collect_params().initialize(
  mx.init.Normal(sigma=1.), ctx=ctx)

loss = gluon.loss.L2Loss()

sgb = gluon.Trainer(net.collect_params(),
                    'sgd',
                    {'learning_rate': lr})
```

```
# train!
train_d = gluon.data.DataLoader(
  mnist_train,
  batch_size=batch_size, shuffle=True)

for e in range(epochs):
  for i, (data, label) in enumerate(train_d):
    with autograd.record():
      output = net(data)
      _loss = loss(output, label)
    _loss.backward()
    sgd.step(batch_size)

  print('curr loss:',
        nd.mean(_loss).assscalar())
```

Example: your own layer

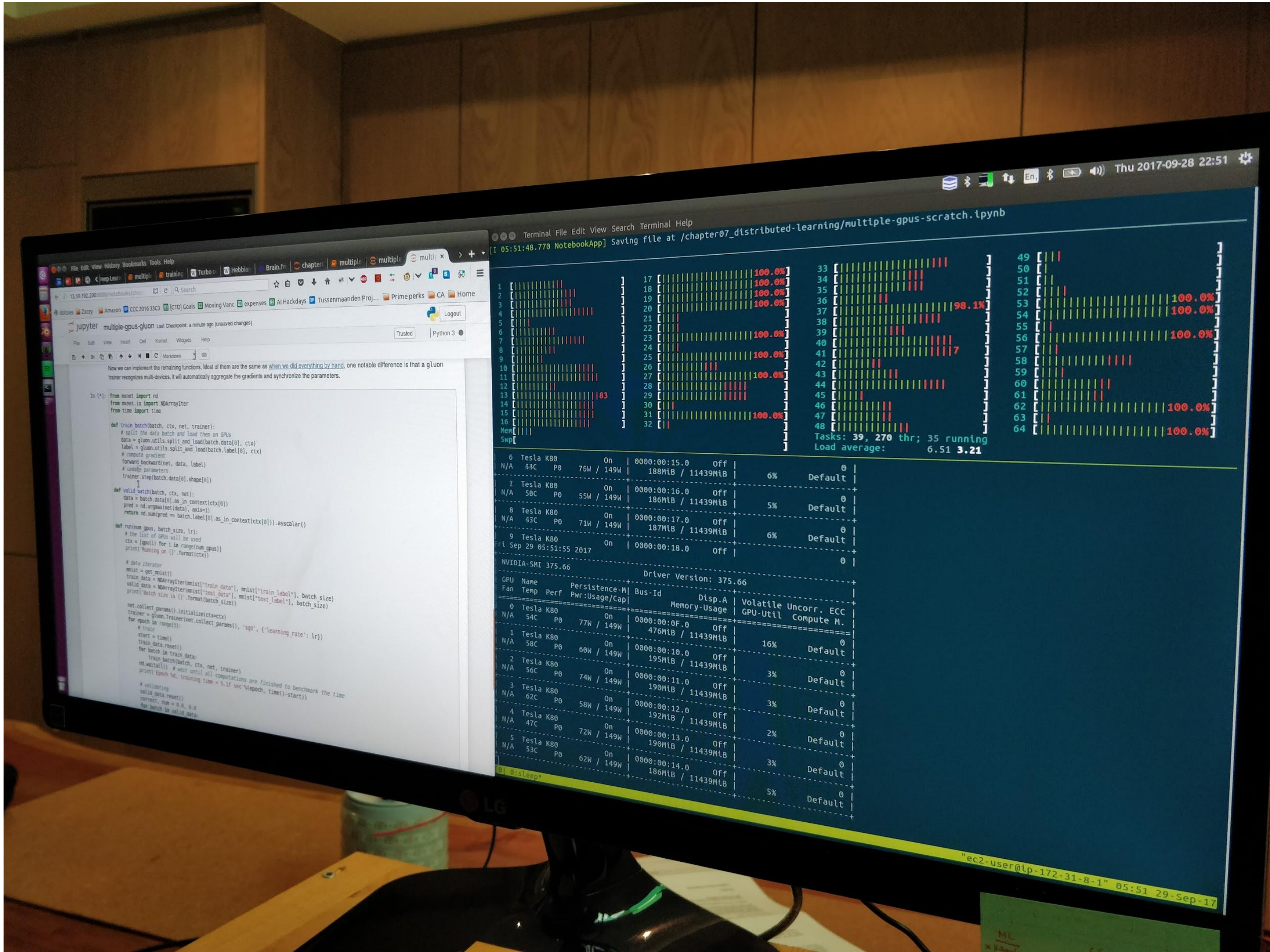
```
class MyDense(HybridBlock):
    def __init__(self, units, in_units=0, **kwargs):
        super(MyDense, self).__init__(**kwargs)
        with self.name_scope():
            self.units = units
            # gluon.Parameter with 0-valued shape elements means: will be filled in later
            self._in_units = in_units
            # Add parameters to internal ParameterDict, indicating the desired shape
            self.weight = self.params.get(
                'weight', init=mx.init.Xavier(magnitude=2.24),
                shape=(in_units, units))
            self.bias = self.params.get('bias', shape=(units,))

    def hybrid_forward(self, F, x):  # F will be mxnet.nd or mxnet.sym
        with x.context:
            linear = F.dot(x, self.weight.data()) + self.bias.data()
            activation = F.maximum(linear, 0)
            return activation
    # backward will be generated automatically
```

Fast

● running as C++ extension: highly optimised	10-100x
● parallelised: smart scheduler models dependencies	1-16x
● multiple GPU support, multiple hosts support	10x 10x 10x
● imperative (ndarray): faster coding, caches results	
symbolic (sym): also faster execution with optimised graph	2x
	------------ +
	1.000.000 x

Parallel

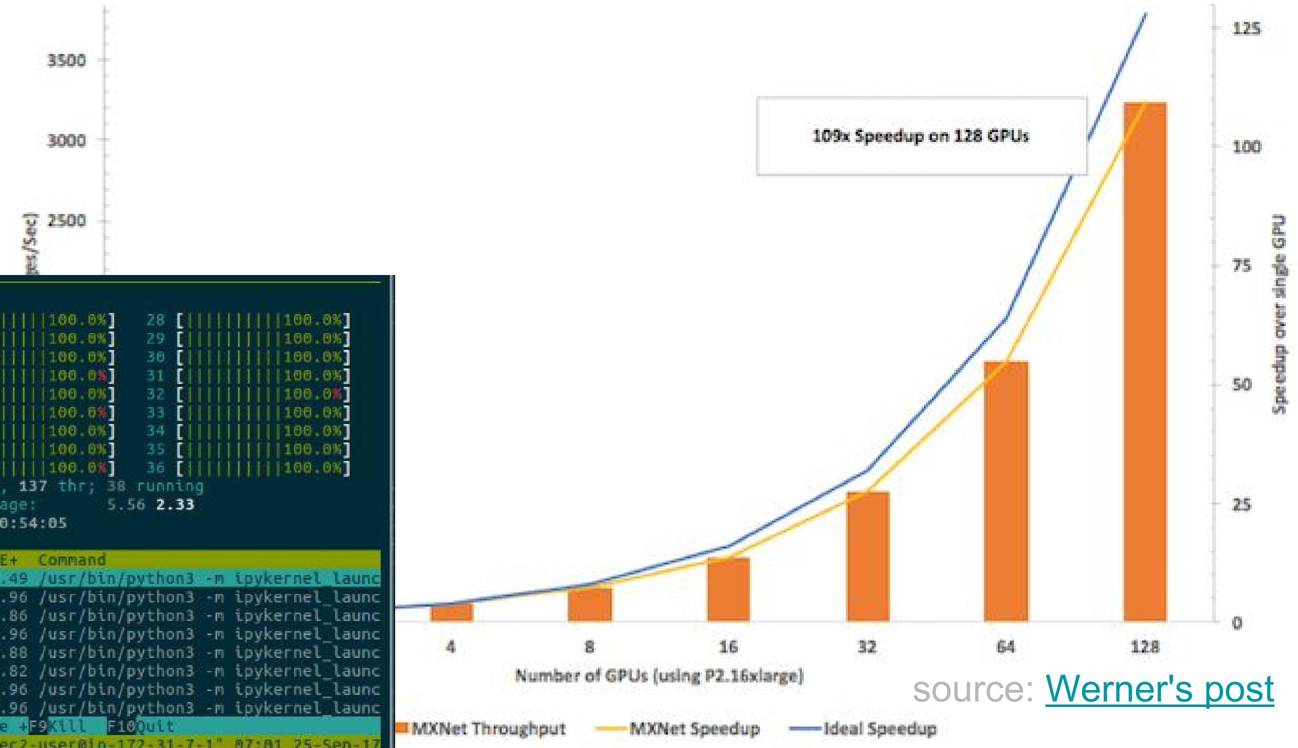
- Multiple CPUs
- Multiple GPUs
- Multiple hosts

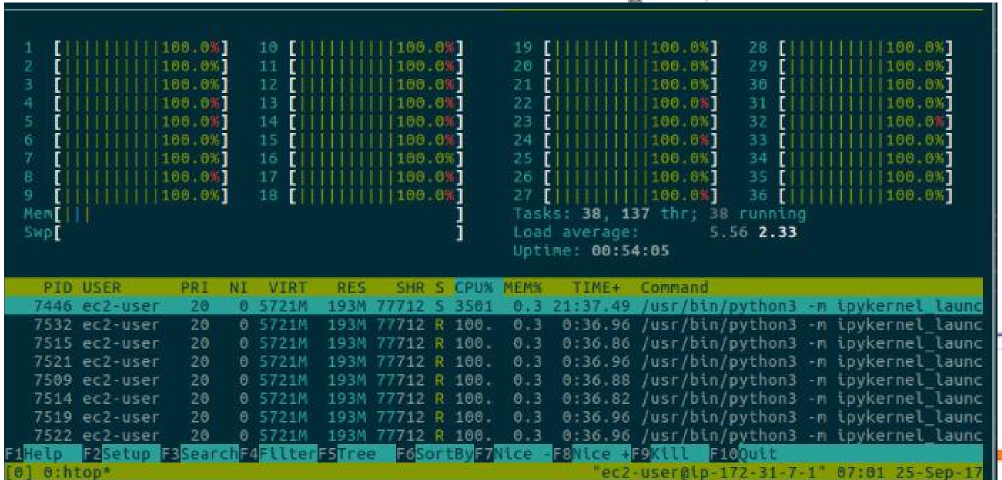


source: my laptop

Parallel

- ctx, copyto, as_in_context, etc
- exmpl using multiple CPU / GPU
- kvstore for networking
- Trainer supports all this





source: Werner's post

Resources

- The Straight Dope
- MXNet tutorials
- Follow-up presentation?