async-computation

April 9, 2019

1 Asynchronous Computing

```
In [1]: from mxnet import autograd, gluon, nd
        from mxnet.gluon import loss as gloss, nn
        import os
        import subprocess
        import time

        class Benchmark():
            def __init__(self, prefix=None):
                self.prefix = prefix + ' ' if prefix else ''

            def __enter__(self):
                self.start = time.time()

            def __exit__(self, *args):
                print('%stime: %.4f sec' % (self.prefix, time.time() - self.start))
```

1.1 Asynchronous Execution in MXNet

```
In [2]: with Benchmark('Workloads are queued.'):
            x = nd.random.uniform(shape=(2000, 2000))
            y = nd.dot(x, x).sum()

        with Benchmark('Workloads are finished.'):
            print('sum =', y)
```

```
Workloads are queued. time: 0.0006 sec
('sum =',
[2.0003661e+09]
<NDArray 1 @cpu(0)>)
Workloads are finished. time: 0.1748 sec
```

1.2 Use Synchronization Functions to Allow the Front-end to Wait

```
In [3]: with Benchmark():
            y = nd.dot(x, x)
```

```
            y.wait_to_read()

        with Benchmark():
            y = nd.dot(x, x)
            z = nd.dot(x, x)
            nd.waitall()

        with Benchmark():
            y = nd.dot(x, x)
            y.asnumpy()

        with Benchmark():
            y = nd.dot(x, x)
            y.norm().asscalar()
```

```
time: 0.0649 sec
time: 0.1287 sec
time: 0.0670 sec
time: 0.2124 sec
```

1.3 Using Asynchronous Programming to Improve Computing Performance

```
In [4]: with Benchmark('synchronous.'):
            for _ in range(1000):
                y = x + 1
                y.wait_to_read()

        with Benchmark('asynchronous.'):
            for _ in range(1000):
                y = x + 1
            nd.waitall()
```

```
synchronous. time: 0.7633 sec
asynchronous. time: 0.5544 sec
```

1.4 The Impact of Asynchronous Programming on Memory

```
In [5]: def data_iter():
            start = time.time()
            num_batches, batch_size = 100, 1024
            for i in range(num_batches):
                X = nd.random.normal(shape=(batch_size, 512))
                y = nd.ones((batch_size,))
                yield X, y
                if (i + 1) % 50 == 0:
                    print('batch %d, time %f sec' % (i + 1, time.time() - start))
```

1.4.1 Train a MLP

```
In [6]: net = nn.Sequential()
        net.add(nn.Dense(2048, activation='relu'),
                nn.Dense(512, activation='relu'),
                nn.Dense(1))
        net.initialize()
        trainer = gluon.Trainer(net.collect_params(), 'sgd', {'learning_rate': 0.005})
        loss = gloss.L2Loss()
        for X, y in data_iter():
            break
        loss(y, net(X)).wait_to_read()
```

1.4.2 Check Memory Usage

```
In [7]: def get_mem():
            res = subprocess.check_output(['ps', 'u', '-p', str(os.getpid())])
            return int(str(res).split()[15]) / 1e3
```

1.4.3 Synchronize in each Batch

```
In [8]: l_sum, mem = 0, get_mem()
        for X, y in data_iter():
            with autograd.record():
                l = loss(y, net(X))
            l_sum += l.mean().asscalar()  # Use of the Asscalar synchronization function.
            l.backward()
            trainer.step(X.shape[0])
        nd.waitall()
        print('increased memory: %f MB' % (get_mem() - mem))
```

```
batch 50, time 2.926678 sec
batch 100, time 5.887035 sec
increased memory: 0.256000 MB
```

1.4.4 No Synchronization between Batches

```
In [9]: mem = get_mem()
        for X, y in data_iter():
            with autograd.record():
                l = loss(y, net(X))
            l.backward()
            trainer.step(x.shape[0])
        nd.waitall()
        print('increased memory: %f MB' % (get_mem() - mem))
```

```
batch 50, time 0.071198 sec
batch 100, time 0.142361 sec
```

```
increased memory: 196.608000 MB
```