multiple-gpus

April 9, 2019

1 Multi-GPU Computation with Data Parallelism

```
In [1]: import d2l
        import mxnet as mx
        from mxnet import autograd, nd
        from mxnet.gluon import loss as gloss
        import time

        !nvidia-smi
```

```
Sat Jan 19 00:41:21 2019
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 396.37                 Driver Version: 396.37                    |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla M60           Off  | 00000000:00:1D.0 Off |                    0 |
| N/A   31C    P0    38W / 150W |   2639MiB /  7618MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla M60           Off  | 00000000:00:1E.0 Off |                    0 |
| N/A   33C    P8    13W / 150W |     11MiB /  7618MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
+-----------------------------------------------------------------------------+
```

1.1 Define the Model: LeNet

```
In [3]: scale = 0.01
        W1 = nd.random.normal(scale=scale, shape=(20, 1, 3, 3))
        b1 = nd.zeros(shape=20)
        W2 = nd.random.normal(scale=scale, shape=(50, 20, 5, 5))
```

```
b2 = nd.zeros(shape=50)
W3 = nd.random.normal(scale=scale, shape=(800, 128))
b3 = nd.zeros(shape=128)
W4 = nd.random.normal(scale=scale, shape=(128, 10))
b4 = nd.zeros(shape=10)
params = [W1, b1, W2, b2, W3, b3, W4, b4]

def lenet(X, params):
    h1_conv = nd.Convolution(data=X, weight=params[0], bias=params[1],
                             kernel=(3, 3), num_filter=20)
    h1_activation = nd.relu(h1_conv)
    h1 = nd.Pooling(data=h1_activation, pool_type='avg', kernel=(2, 2),
                    stride=(2, 2))
    h2_conv = nd.Convolution(data=h1, weight=params[2], bias=params[3],
                             kernel=(5, 5), num_filter=50)
    h2_activation = nd.relu(h2_conv)
    h2 = nd.Pooling(data=h2_activation, pool_type='avg', kernel=(2, 2),
                    stride=(2, 2))
    h2 = nd.flatten(h2)
    h3_linear = nd.dot(h2, params[4]) + params[5]
    h3 = nd.relu(h3_linear)
    y_hat = nd.dot(h3, params[6]) + params[7]
    return y_hat

loss = gloss.SoftmaxCrossEntropyLoss()
```

1.2 Copy Parameter to a Device

In [4]:
```
def get_params(params, ctx):
    new_params = [p.copyto(ctx) for p in params]
    for p in new_params:
        p.attach_grad()
    return new_params

new_params = get_params(params, mx.gpu(0))
print('b1 weight:', new_params[1])
print('b1 grad:', new_params[1].grad)
```

1.3 Sum Over All Devices and then Broadcast

In [6]:
```
def allreduce(data):
    for i in range(1, len(data)):
        data[0][:] += data[i].copyto(data[0].context)
    for i in range(1, len(data)):
        data[0].copyto(data[i])

data = [nd.ones((1, 2), ctx=mx.gpu(i)) * (i + 1) for i in range(2)]
print('before allreduce:', data)
```

```
allreduce(data)
print('after allreduce:', data)
```

1.4 Split a Data Batch into Each GPUs

In [8]:
```
def split_and_load(data, ctx):
    n, k = data.shape[0], len(ctx)
    m = n // k  # For simplicity, we assume the data is divisible.
    assert m * k == n, '# examples is not divided by # devices.'
    return [data[i * m: (i + 1) * m].as_in_context(ctx[i]) for i in range(k)]

batch = nd.arange(24).reshape((6, 4))
ctx = [mx.gpu(0), mx.gpu(1)]
splitted = split_and_load(batch, ctx)
print('input: ', batch)
print('load into', ctx)
print('output:', splitted)
```

1.5 Multi-GPU Training on a Single Mini-batch

In [10]:
```
def train_batch(X, y, gpu_params, ctx, lr):
    # When ctx contains multiple GPUs, mini-batches of data instances are divided and
    gpu_Xs, gpu_ys = split_and_load(X, ctx), split_and_load(y, ctx)
    with autograd.record():  # Loss is calculated separately on each GPU.
        ls = [loss(lenet(gpu_X, gpu_W), gpu_y)
              for gpu_X, gpu_y, gpu_W in zip(gpu_Xs, gpu_ys, gpu_params)]
    for l in ls:  # Back Propagation is performed separately on each GPU.
        l.backward()
    # Add up all the gradients from each GPU and then broadcast them to all the GPUs.
    for i in range(len(gpu_params[0])):
        allreduce([gpu_params[c][i].grad for c in range(len(ctx))])
    for param in gpu_params:  # The model parameters are updated separately on each G
        d2l.sgd(param, lr, X.shape[0])  # Here, we use a full-size batch.
```

1.6 Training Functions

In [11]:
```
def train(num_gpus, batch_size, lr):
    train_iter, test_iter = d2l.load_data_fashion_mnist(batch_size)
    ctx = [mx.gpu(i) for i in range(num_gpus)]
    print('running on:', ctx)
    # Copy model parameters to num_gpus GPUs.
    gpu_params = [get_params(params, c) for c in ctx]
    for epoch in range(4):
        start = time.time()
        for X, y in train_iter:
            # Perform multi-GPU training for a single mini-batch.
            train_batch(X, y, gpu_params, ctx, lr)
            nd.waitall()
```

```
        train_time = time.time() - start

        def net(x):  # Verify the model on GPU 0.
            return lenet(x, gpu_params[0])

        test_acc = d2l.evaluate_accuracy(test_iter, net, ctx[0])
        print('epoch %d, time: %.1f sec, test acc: %.2f'
              % (epoch + 1, train_time, test_acc))
```

1.7 Multi-GPU Training Experiment

```
In [12]: train(num_gpus=1, batch_size=256, lr=0.2)
```

```
running on: [gpu(0)]
epoch 1, time: 2.7 sec, test acc: 0.10
epoch 2, time: 2.2 sec, test acc: 0.66
epoch 3, time: 2.2 sec, test acc: 0.75
epoch 4, time: 2.2 sec, test acc: 0.71
```

```
In [13]: train(num_gpus=2, batch_size=256, lr=0.2)
```

```
running on: [gpu(0), gpu(1)]
epoch 1, time: 2.5 sec, test acc: 0.10
epoch 2, time: 2.2 sec, test acc: 0.61
epoch 3, time: 2.2 sec, test acc: 0.75
epoch 4, time: 2.2 sec, test acc: 0.76
```