multiple-gpus-gluon

April 9, 2019

1 Gluon Implementation for Multi-GPU Computation

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

1.1 Create Model

```
In [2]: def resnet18(num_classes):  # This function is saved in the d2l package for future use
            def resnet_block(num_channels, num_residuals, first_block=False):
                blk = nn.Sequential()
                for i in range(num_residuals):
                    if i == 0 and not first_block:
                        blk.add(d2l.Residual(
                            num_channels, use_1x1conv=True, strides=2))
                    else:
                        blk.add(d2l.Residual(num_channels))
                return blk

            net = nn.Sequential()
            # This model uses a smaller convolution kernel, stride, and padding and removes th
            net.add(nn.Conv2D(64, kernel_size=3, strides=1, padding=1),
                    nn.BatchNorm(), nn.Activation('relu'))
            net.add(resnet_block(64, 2, first_block=True),
                    resnet_block(128, 2),
                    resnet_block(256, 2),
                    resnet_block(512, 2))
            net.add(nn.GlobalAvgPool2D(), nn.Dense(num_classes))
            return net

        net = resnet18(10)
```

1.2 Initialize Parameters on Multiple GPUs

```
In [3]: ctx = [mx.gpu(0), mx.gpu(1)]
        net.initialize(init=init.Normal(sigma=0.01), ctx=ctx)
```

1.3 Split a Data Batch into Each GPUs

```
In [4]: x = nd.random.uniform(shape=(4, 1, 28, 28))
        gpu_x = gutils.split_and_load(x, ctx)
        net(gpu_x[0]), net(gpu_x[1])
```

```
Out[4]: (
         [[ 5.4814927e-06 -8.3370861e-07 -1.6316785e-06 -6.3674065e-07
           -3.8216158e-06 -2.3514044e-06 -2.5469603e-06 -9.4783900e-08
           -6.9033780e-07  2.5756226e-06]
          [ 5.4710858e-06 -9.4246485e-07 -1.0494068e-06  9.8082182e-08
           -3.3251827e-06 -2.4862920e-06 -3.3642798e-06  1.0455926e-07
           -6.1001288e-07  2.0327857e-06]]
         <NDArray 2x10 @gpu(0)>,
         [[ 5.61763409e-06 -1.28376018e-06 -1.46055379e-06  1.83030124e-07
           -3.55116526e-06 -2.43710247e-06 -3.57317913e-06 -3.09749055e-07
           -1.10165593e-06  1.89098898e-06]
          [ 5.14187059e-06 -1.37299276e-06 -1.15200976e-06  1.15073135e-07
           -3.73728039e-06 -2.82897145e-06 -3.64771859e-06  1.57815123e-07
           -6.07330094e-07  1.97120244e-06]]
         <NDArray 2x10 @gpu(1)>)
```

1.4 Access Parameters on Given Device

```
In [5]: weight = net[0].params.get('weight')

        try:
            weight.data()
        except RuntimeError:
            print('not initialized on', mx.cpu())
        weight.data(ctx[0])[0], weight.data(ctx[1])[0]
```

```
not initialized on cpu(0)
```

```
Out[5]: (
         [[[-0.01473444 -0.01073093 -0.01042483]
           [-0.01327885 -0.01474966 -0.00524142]
           [ 0.01266256  0.00895064 -0.00601594]]]
         <NDArray 1x3x3 @gpu(0)>,
         [[[-0.01473444 -0.01073093 -0.01042483]
           [-0.01327885 -0.01474966 -0.00524142]
           [ 0.01266256  0.00895064 -0.00601594]]]
         <NDArray 1x3x3 @gpu(1)>)
```

1.5 Training Functions

```
In [6]: 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)
            net.initialize(init=init.Normal(sigma=0.01), ctx=ctx, force_reinit=True)
            trainer = gluon.Trainer(
                net.collect_params(), 'sgd', {'learning_rate': lr})
            loss = gloss.SoftmaxCrossEntropyLoss()
            for epoch in range(4):
                start = time.time()
                for X, y in train_iter:
                    gpu_Xs = gutils.split_and_load(X, ctx)
                    gpu_ys = gutils.split_and_load(y, ctx)
                    with autograd.record():
                        ls = [loss(net(gpu_X), gpu_y)
                              for gpu_X, gpu_y in zip(gpu_Xs, gpu_ys)]
                    for l in ls:
                        l.backward()
                    trainer.step(batch_size)
                nd.waitall()
                train_time = time.time() - start
                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))
```

1.6 Multi-GPU Training Experiment

```
In [7]: train(num_gpus=1, batch_size=256, lr=0.1)

running on: [gpu(0)]
epoch 1, time: 63.7 sec, test acc 0.88
epoch 2, time: 59.3 sec, test acc 0.91
epoch 3, time: 59.4 sec, test acc 0.90
epoch 4, time: 59.5 sec, test acc 0.93


In [8]: train(num_gpus=2, batch_size=512, lr=0.2)

running on: [gpu(0), gpu(1)]
epoch 1, time: 31.5 sec, test acc 0.75
epoch 2, time: 30.5 sec, test acc 0.85
epoch 3, time: 30.4 sec, test acc 0.90
epoch 4, time: 30.5 sec, test acc 0.91
```