**Chainer：https://docs.chainer.org/en/stable/index.html**

Chainer是由intel开发的深度学习框架，与cupy结合使用。Cupy可以理解为是一个运用GPU的numpy工具包。Chainer使用cupy的memory pool来避免数据的重复读写，从而提高效率。

**安装：**

对于这里的计算平台，一般要明确cupy版本，我使用的是cuda9.2，因此我安装了cupy-cuda92与之对应。否则可能安装失败（直接pip install cupy可能是个坑，踩了很久），安装的主要几个注意事项：

pip install -U setuptools pip  
pip install cupy-cuda92==6.6.0

pip install chainer==6.5.0  
pip install chainercv==0.13.1

为了保证兼容性，cupy的版本不能过高于chainer

**Chainer的多GPU使用：**

在cpu和gpu中数据转换使用[**chainer.backends.cuda.to\_gpu()**](https://docs.chainer.org/en/stable/reference/generated/chainer.backends.cuda.to_gpu.html#chainer.backends.cuda.to_gpu)函数，和[**chainer.backends.cuda.to\_cpu()**](https://docs.chainer.org/en/stable/reference/generated/chainer.backends.cuda.to_gpu.html#chainer.backends.cuda.to_gpu)

Chainer的一些methods：

* Chainer.functions.copy(array, 1):把arraycopy到gpu1上，如果瞎写1234device会报错，会自动检测GPU数量， GPU从0开始计算
* Chainer中的网络结构都是一个link object, 可以通过to\_gpu等操作实现device的转换

**Model-parallel Computation on Multiple GPUs：**

**Link：https://docs.chainer.org/en/stable/guides/gpu.html**

机器学习的并行大致分为两种，model-parallel是指在模型内部进行并行运算，而data-parallel意味着对数据进行切片。

对于model parallel的模型相当于AlexNet的做法，走两条路，然后在某些节点进行数据交互，获得最终输出，如图例：

(GPU0) input --+--> mlp1 --+--> mlp2 --+--> output

| | |

(GPU1) +--> mlp1 --+--> mlp2 --+

对于data parallel的模型，我们采取复制模型，分别更新的方法

复制模型：

model\_1 = model\_0.copy()

model\_0.to\_gpu(0)

model\_1.to\_gpu(1)

设置optimizer：

Optimizer = optimizers.SGD（）

Optimizer.setup(model\_0)

采用类似以下的loop实现数据并行

batchsize = 100

datasize = len(x\_train)

**for** epoch **in** range(20):

print('epoch *%d*' % epoch)

indexes = np.random.permutation(datasize)

**for** i **in** range(0, datasize, batchsize):

x\_batch = x\_train[indexes[i : i + batchsize]]

y\_batch = y\_train[indexes[i : i + batchsize]]

x0 = Variable(cuda.to\_gpu(x\_batch[:batchsize//2], 0))

t0 = Variable(cuda.to\_gpu(y\_batch[:batchsize//2], 0))

x1 = Variable(cuda.to\_gpu(x\_batch[batchsize//2:], 1))

t1 = Variable(cuda.to\_gpu(y\_batch[batchsize//2:], 1))

loss\_0 = model\_0(x0, t0)

loss\_1 = model\_1(x1, t1)

model\_0.cleargrads()

model\_1.cleargrads()

loss\_0.backward()

loss\_1.backward()

model\_0.addgrads(model\_1)

optimizer.update()

model\_1.copyparams(model\_0)

事实上chainer也有自带的两种并行方案:

Chainer.training.updaters.ParallelUpdater以及Chainer.training.updaters.MultiprocessParallelUpdator

**Chainer必须保持模型所在的设备和索要运算的变量是一致的**

对于chainer的model使用 model.device可以调出来设备名称

如果想看某个变量是什么设备上的可以使用cuda.get\_device\_from\_array(array)