Apache MXNet Cheat Sheet

Sunil Mallya: smallya@amazon.com

Multilayer Perceptron (MLP)

```
net = mx.sym.Variable('data')
net = mx.sym.FullyConnected(net, name='fc1', num_hidden=64)
net = mx.sym.Activation(net, name='relu1', act_type="relu")
net = mx.sym.FullyConnected(net, name='fc2', num_hidden=10)
mlp = mx.sym.SoftmaxOutput(net, name='softmax')
```

NDArray Iterator

```
mx.io.NDArrayIter(data=np.zeros((1200, 3, 224, 224), dtype='float32'),
        label={'annot': np.zeros((1200, 80), dtype='int8'),
        'label': np.zeros((1200, 1), dtype='int8')}, batch_size=10)
```

CSV Iterator

```
csv_iter = mx.io.CSVIter(data_csv='data.csv',
        data_shape = (3,), batch_size = 3)
```

Linear Regression

```
X = mx.sym.Variable('data')
Y = mx.symbol.Variable('lin_reg_label')
fully_connected_layer  = mx.sym.FullyConnected(data=X,
        name='fc1', num_hidden = 1)
lro = mx.sym.LinearRegressionOutput(
        data=fully_connected_layer, label=Y, name="lro")
```

Logistic Regression

```
data = mx.sym.Variable("data")
target = mx.sym.Variable("target")
fc = mx.sym.FullyConnected(data=data,
        num_hidden=1, name='fc')
pred = mx.sym.LogisticRegressionOutput(
        data=fc, label=target)
```

Save, Checkpoint, Load and Predict

```
# save
prefix = 'mymodel'
iteration = 100
model.save(prefix, iteration)

# periodic checkpoint
mod.fit(X, Y, eval_metric="acc", ...
        epoch_end_callback=mx.callback.save.checkpoint("mymodel")

# load model
sym, arg_params, aux_params = mx.model.load_checkpoint(prefix, epoch_num)
mod = mx.mod.Module(symbol=sym, context=mx.gpu())
mod.bind([('data', (BATCH_SIZE, NDARRAY_SHAPE))], for_training=False)
mod.set_params(arg_params=arg_params, aux_params=aux_params)

# Prediction
Batch = collections.namedtuple('Batch', ['data'])
mod.forward(Batch(batch), is_train=False)
out = mod.get_outputs()[0].asnumpy()
```

Lenet (CNN example)

```
data = mx.symbol.Variable('data')
conv1 = mx.sym.Convolution(data=data, kernel=(5,5),
  num_filter=20)
tanh1 = mx.sym.Activation(data=conv1, act_type="tanh")
pool1 = mx.sym.Pooling(data=tanh1, pool_type="max", kernel
  =(2,2), stride=(2,2))
conv2 = mx.sym.Convolution(data=pool1, kernel=(5,5),
  num_filter=50)
tanh2 = mx.sym.Activation(data=conv2, act_type="tanh")
pool2 = mx.sym.Pooling(data=tanh2, pool_type="max", kernel
  =(2,2), stride=(2,2))
flatten = mx.sym.Flatten(data=pool2)
fc1 = mx.symbol.FullyConnected(data=flatten, num_hidden=500)
tanh3 = mx.sym.Activation(data=fc1, act_type="tanh")
fc2 = mx.sym.FullyConnected(data=tanh3, num_hidden=10)
lenet = mx.sym.SoftmaxOutput(data=fc2, name='softmax')
mx.viz.plot_network(symbol=lenet, shape=shape)
```

Image Iterator

```
data_iter = mx.io.ImageRecordIter(path_imgrec="data.rec",
            data_shape=(3, 227, 227), batch_size=4)
```

S3 Image Iterator

```
#Append USE_S3=1 to config.mk before building MXNet
s3_data_iter = mx.io.ImageRecordIter(
    path_imgrec="s3://bucketname/train/data.rec",
    data_shape=(3, 227, 227), batch_size=4, resize=256)
```

Scoring

```
mod.score(data.get_iter(batch_size), ['mse', 'acc'])
```

Convolution Factory

```
def ConvFactory(data, num_filter, kernel, stride=(1,1), pad=(0, 0), name=None, suffix=''):
    conv = mx.symbol.Convolution(data=data, num_filter=num_filter, kernel=kernel,
                                 stride=stride, pad=pad, name='conv_%s%s' %(name, suffix))
    bn = mx.symbol.BatchNorm(data=conv, name='bn_%s%s' %(name, suffix))
    act = mx.symbol.Activation(data=bn, act_type='relu', name='relu_%s%s' %(name, suffix))
    return act
prev = mx.symbol.Variable(name="Previos Output")
conv_comp = ConvFactory(data=prev, num_filter=64, kernel=(7,7), stride=(2, 2))
```

Multiple GPUs (one-liner)

```
devices = [mx.gpu(i) for i in range(num_device)]
mod = mx.module.Module(context=devices, ...)
```

Module

```
mod = mx.mod.Module(symbol=net, context=mx.cpu(),
                    data_names=['data'],
                    label_names=['softmax_label'])
```

Fine-Tuning

```
sym, arg_params, aux_params = mx.model.load_checkpoint(prefix, epoch)
all_layers = sym.get_internals()
net = all_layers[layer_name+'_output']
net = mx.symbol.FullyConnected(data=net, num_hidden=num_classes, name='fc')
net = mx.symbol.SoftmaxOutput(data=net, name='softmax')
new_args = dict({k:arg_params[k] for k in arg_params if 'fc' not in k})
```

Learing rate scheduler

```
lr_sch = mx.lr_scheduler.FactorScheduler(step=100,
  factor=0.9)
mod.init_optimizer(optimizer='sgd',
          optimizer_params=(('learning_rate', 0.1),
           lr_scheduler', lr_sch)))
```

RNN: LSTM

```
data = mx.sym.var("data")
target = mx.sym.var("target")
lstm1 = mx.rnn.FusedRNNCell(num_hidden=1,
        mode="lstm", prefix="lstm1_")
L1, L1_states = lstm1.unroll(length=seq_len, inputs=data,
        merge_outputs=True, layout="TNC")
pred = mx.sym.LinearRegressionOutput(data=L1, label=target)
```

Module Low Level API

```
mod = mx.mod.Module(symbol=net)
train_iter = data.get_iter(batch_size)
mod.bind(data_shapes=train_iter.provide_data,
         label_shapes=train_iter.provide_label)
mod.init_params(initializer=mx.init.Xavier(magnitude=2))
mod.init_optimizer(optimizer='sgd',
         optimizer_params=(('learning_rate', 0.1), ))
metric = mx.metric.create('acc')
for batch in train_iter:
    mod.forward(batch, is_train=True)
    mod.update_metric(metric, batch.label)
    mod.backward()
    mod.update()
```