MXNet Cheat Sheet

A quick reference guide for Apache MXNet, covering essential concepts, modules, and operations for building and training neural networks.

Core Concepts

NDArray

The fundamental data structure in MXNet, similar to NumPy's ndarray. It represents multi-dimensional arrays.

- **Creation:** `mx.nd.array(data)`
- **Shape:** `ndarray.shape`
- **Context:** `mx.cpu()` or `mx.gpu(0)` to specify device.

Example:

```
import mxnet as mx

# Create an NDArray on CPU
a = mx.nd.array([1, 2, 3], ctx=mx.cpu())
print(a)

# Create an NDArray on GPU (if available)
b = mx.nd.array([4, 5, 6], ctx=mx.gpu(0) if
mx.context.num_gpus() else mx.cpu())
print(b)
```

Symbol

Represents a symbolic expression for defining neural network architectures. Symbols are used to define the computation graph.

- **Defining Operations:** Use operators like `mx.sym.Convolution`, `mx.sym.Activation`, etc.
- **Creating a Network:** Combine symbols to create a computational graph.

Example:

```
import mxnet as mx

# Define a simple neural network
data = mx.sym.Variable('data')
fc1  = mx.sym.FullyConnected(data=data,
num_hidden=128)
act1 = mx.sym.Activation(data=fc1,
act_type='relu')
fc2  = mx.sym.FullyConnected(data=act1,
num_hidden=10)
softmax = mx.sym.SoftmaxOutput(data=fc2,
name='softmax')

# Print the symbol graph
print(softmax.list_arguments())
```

Context

Specifies the device (CPU or GPU) on which the computation will be performed.

- `mx.cpu()`: Use CPU.
- `mx.gpu(device_id)`: Use GPU with the specified ID.
- Important for running your models on GPUs for faster training.

Example:

```
import mxnet as mx

# Define context
ctx = mx.gpu() if mx.context.num_gpus() else
mx.cpu()

# Create NDArray in the defined context
a = mx.nd.ones((2, 2), ctx=ctx)
print(a)
```

Neural Network Layers

Convolutional Layers

Used for feature extraction from images.

- `mx.sym.Convolution(data=input, kernel=(height, width), num_filter=num_filters)`
- `num_filter`: Number of output filters.
- `kernel`: Shape of the convolutional kernel.

Example:

```
import mxnet as mx

data = mx.sym.Variable('data')
conv1 = mx.sym.Convolution(data=data, kernel=
(3, 3), num_filter=32)
print(conv1.list_arguments())
```

Pooling Layers

Used for reducing the spatial dimensions of the feature maps.

- `mx.sym.Pooling(data=input, pool_type='max', kernel=(height, width), stride=(stride_height, stride_width))`
- `pool_type`: 'max', 'avg', etc.
- `kernel`: Shape of the pooling kernel.
- `stride`: Stride of the pooling operation.

Example:

```
import mxnet as mx

data = mx.sym.Variable('data')
pool1 = mx.sym.Pooling(data=data,
pool_type='max', kernel=(2, 2), stride=(2, 2))
print(pool1.list_arguments())
```

Fully Connected Layers

Also known as dense layers, used for classification.

- `mx.sym.FullyConnected(data=input, num_hidden=num_neurons)`
- `num_hidden`: Number of neurons in the layer.

Example:

```
import mxnet as mx

data = mx.sym.Variable('data')
fc1 = mx.sym.FullyConnected(data=data,
num_hidden=128)
print(fc1.list_arguments())
```

Activation Functions

Apply a non-linear transformation to the output of a layer.

- `mx.sym.Activation(data=input, act_type='relu')`
- `act_type`: 'relu', 'sigmoid', 'tanh', etc.

Example:

```
import mxnet as mx

data = mx.sym.Variable('data')
act1 = mx.sym.Activation(data=data,
act_type='relu')
print(act1.list_arguments())
```

Training and Evaluation

Data Loading

Loading data for training.

- `mx.io.NDArrayIter(data, label, batch_size)`: Creates an iterator from NDArrays.
- `mx.io.ImageRecordIter(path_imgrec, data_shape, batch_size)`: Loads data from image record files.

Example:

```
import mxnet as mx
import numpy as np

# Create dummy data
data = np.random.rand(100, 1, 28, 28)
label = np.random.randint(0, 10, (100,))

# Create data iterator
data_iter = mx.io.NDArrayIter(data, label,
batch_size=32)
```

Optimizer

Algorithm to update the weights of the network during training.

- `mx.optimizer.SGD(learning_rate=lr, momentum=mu, wd=wd)`: Stochastic Gradient Descent.
- `mx.optimizer.Adam(learning_rate=lr, beta1=0.9, beta2=0.999)`: Adam optimizer.

Example:

```
import mxnet as mx

# Create an SGD optimizer
optimizer =
mx.optimizer.SGD(learning_rate=0.01,
momentum=0.9, wd=0.0001)
```

Metrics

Used to evaluate the performance of the model.

- `mx.metric.Accuracy()`: Calculates accuracy.
- `mx.metric.CrossEntropy()`: Calculates cross-entropy loss.

Example:

```
import mxnet as mx

# Create an accuracy metric
metric = mx.metric.Accuracy()
```

Model Training

Training the model using the defined data iterator, symbol, optimizer, and metric.

- `mx.mod.Module(symbol, context, data_names, label_names)`: Creates a module for training.
- `module.fit(data_iter, eval_data=val_iter, optimizer=optimizer, eval_metric=metric, num_epoch=epochs)`: Trains the module.

Example:

```
import mxnet as mx
import numpy as np

# Dummy data and label
data = np.random.rand(100, 1, 28, 28)
label = np.random.randint(0, 10, (100,))
data_iter = mx.io.NDArrayIter(data, label,
batch_size=32)

# Define model
data = mx.sym.Variable('data')
fc1  = mx.sym.FullyConnected(data=data,
num_hidden=10)
softmax = mx.sym.SoftmaxOutput(data=fc1,
name='softmax')

# Create a module
module = mx.mod.Module(symbol=softmax,
context=mx.cpu(), data_names=('data',),
label_names=('softmax_label',))
module.bind(data_shapes=data_iter.provide_data
, label_shapes=data_iter.provide_label)
module.initialize(mx.init.Xavier(magnitude=2.2
4))

# Set optimizer
optimizer =
mx.optimizer.SGD(learning_rate=0.01,
momentum=0.9, wd=0.0001)
module.fit(data_iter, optimizer=optimizer,
num_epoch=10)
```

Gluon API

Gluon Basics

A high-level API for building neural networks in MXNet. Provides a more intuitive and flexible way to define, train, and evaluate models.

- **`gluon.nn`**: A set of pre-defined layers.
- **`gluon.data`**: Data loading utilities.
- **`gluon.Trainer`**: Training loop manager.

Example:

```
from mxnet import gluon

# Define a sequential neural network
net = gluon.nn.Sequential()
with net.name_scope():
    net.add(gluon.nn.Dense(128,
activation='relu'))
    net.add(gluon.nn.Dense(10))
print(net)
```

Defining a Network with Gluon

Using `gluon.nn.Sequential` and `gluon.nn.Dense` to define neural network architectures easily.

- `gluon.nn.Sequential()`: A sequential container for layers.
- `gluon.nn.Dense(units=num_neurons, activation=activation_function)`: A fully connected layer.

Example:

```
from mxnet import gluon, init
from mxnet.gluon import nn

# Define a network
net = nn.Sequential()
with net.name_scope():
    net.add(nn.Dense(128, activation='relu'))
    net.add(nn.Dense(10))

# Initialize parameters
net.initialize(init.Xavier())
```

Training with Gluon

Using `gluon.Trainer` to simplify the training process.

- `gluon.Trainer(net.collect_params(), optimizer='sgd', optimizer_params={'learning_rate': lr})`
- `Trainer.step(batch_size)` to update the model parameters.

Example:

```
from mxnet import gluon, autograd, nd
from mxnet.gluon import loss as gloss

# Loss function
loss_fn = gloss.SoftmaxCrossEntropyLoss()

# Trainer
trainer = gluon.Trainer(net.collect_params(),
'sgd', {'learning_rate': 0.01})

# Training loop (simplified)
with autograd.record():
    output = net(data)
    loss = loss_fn(output, label)
loss.backward()
trainer.step(data.shape[0])
```

Data Loading with Gluon

Loading data using `gluon.data.DataLoader` for efficient batching and shuffling.

- `gluon.data.DataLoader(dataset, batch_size, shuffle)`: Creates a data loader.
- `gluon.data.ArrayDataset(data, label)`: Creates a dataset from arrays.

Example:

```
from mxnet import gluon, nd
from mxnet.gluon import data as gdata

# Create dummy data
data = nd.random.normal(0, 1, (100, 784))
label = nd.random.randint(0, 10, (100,))

# Create dataset
dataset = gdata.ArrayDataset(data, label)

# Create data loader
dataloader = gdata.DataLoader(dataset,
batch_size=32, shuffle=True)

# Iterate through the data
for X, y in dataloader:
    print(X.shape, y.shape)
    break
```