# hybridize

April 9, 2019

## 1 A Hybrid of Imperative and Symbolic Programming

Imperative Programming

```
In [1]: def add(a, b):
            return a + b
        def fancy_func(a, b, c, d):
            e = add(a, b)
            f = add(c, d)
            g = add(e, f)
            return g
        fancy_func(1, 2, 3, 4)
```

```
Out[1]: 10
```

Symbolic Programming

```
In [1]: def add_str():
            return '''
        def add(a, b):
            return a + b
        '''
        def fancy_func_str():
            return '''
        def fancy_func(a, b, c, d):
            e = add(a, b)
            f = add(c, d)
            g = add(e, f)
            return g
        '''
        prog = add_str() + fancy_func_str() + '''
        print(fancy_func(1, 2, 3, 4))
        '''
        print(prog)
        y = compile(prog, '', 'exec')
        exec(y)
```

```
def add(a, b):
    return a + b

def fancy_func(a, b, c, d):
    e = add(a, b)
    f = add(c, d)
    g = add(e, f)
    return g

print(fancy_func(1, 2, 3, 4))
```

```
10
```

## 1.1 Construct with HybridSequential

```
In [3]: from mxnet import nd, sym
        from mxnet.gluon import nn
        import time

        def get_net():
            net = nn.HybridSequential()
            net.add(nn.Dense(256, activation='relu'),
                    nn.Dense(128, activation='relu'),
                    nn.Dense(2))
            net.initialize()
            return net

        x = nd.random.normal(shape=(1, 512))
        net = get_net()
        net(x)
```

```
Out[3]:
        [[0.08811308 0.06387277]]
        <NDArray 1x2 @cpu(0)>
```

### 1.1.1 Switch to symbolic execution

```
In [4]: net.hybridize()
        net(x)
```

```
Out[4]:
        [[0.08811308 0.06387277]]
        <NDArray 1x2 @cpu(0)>
```

### 1.1.2 Computing Performance

```
In [5]: def benchmark(net, x):
            start = time.time()
            for i in range(1000):
                _ = net(x)
            nd.waitall()
            return time.time() - start
        net = get_net()
        print('before hybridizing: %.4f sec' % (benchmark(net, x)))
        net.hybridize()
        print('after hybridizing: %.4f sec' % (benchmark(net, x)))
```

```
before hybridizing: 0.2385 sec
after hybridizing: 0.1001 sec
```

### 1.1.3 Get the Symbolic Program

```
In [13]: net.export('my_mlp')
         !head -n20 my_mlp-symbol.json
```

```
{
  "nodes": [
    {
      "op": "null",
      "name": "data",
      "inputs": []
    },
    {
      "op": "null",
      "name": "dense6_weight",
      "attrs": {
        "__dtype__": "0",
        "__lr_mult__": "1.0",
        "__shape__": "(10, 0)",
        "__storage_type__": "0",
        "__wd_mult__": "1.0"
      },
      "inputs": []
    },
    {
```

## 1.2 Construct with HybridBlock

```
In [7]: class HybridNet(nn.HybridBlock):
            def __init__(self, **kwargs):
                super(HybridNet, self).__init__(**kwargs)
```

```
        self.hidden = nn.Dense(10)
        self.output = nn.Dense(2)

    def hybrid_forward(self, F, x):
        print('F: ', F)
        print('x: ', x)
        x = F.relu(self.hidden(x))
        print('hidden: ', x)
        return self.output(x)
```

### 1.2.1 Imperative Execution

```
In [8]: net = HybridNet()
        net.initialize()
        x = nd.random.normal(shape=(1, 4))
        net(x)
```

```
F:  <module 'mxnet.ndarray' from '/Users/muli/miniconda3/lib/python3.7/site-packages/mxnet/nda
x:
[[ 0.02184281 -0.31464806 -0.3336492  -0.6471778 ]]
<NDArray 1x4 @cpu(0)>
hidden:
[[0.         0.02384557 0.         0.01206701 0.         0.02765122
  0.         0.03072213 0.02471942 0.        ]]
<NDArray 1x10 @cpu(0)>
```

```
Out[8]:
        [[-0.00021427 -0.00183663]]
        <NDArray 1x2 @cpu(0)>
```

### 1.2.2 Repeat

```
In [9]: net(x)
```

```
F:  <module 'mxnet.ndarray' from '/Users/muli/miniconda3/lib/python3.7/site-packages/mxnet/nda
x:
[[ 0.02184281 -0.31464806 -0.3336492  -0.6471778 ]]
<NDArray 1x4 @cpu(0)>
hidden:
[[0.         0.02384557 0.         0.01206701 0.         0.02765122
  0.         0.03072213 0.02471942 0.        ]]
<NDArray 1x10 @cpu(0)>
```

```
Out[9]:
        [[-0.00021427 -0.00183663]]
        <NDArray 1x2 @cpu(0)>
```

### 1.2.3 Symbolic Execution

```
In [10]: net.hybridize()
         net(x)
```

```
F:  <module 'mxnet.symbol' from '/Users/muli/miniconda3/lib/python3.7/site-packages/mxnet/symbo
x:  <Symbol data>
hidden:  <Symbol hybridnet0_relu0>
```

```
Out[10]:
         [[-0.00021427 -0.00183663]]
         <NDArray 1x2 @cpu(0)>
```

## 1.3 Repeat

```
In [11]: net(x)
```

```
Out[11]:
         [[-0.00021427 -0.00183663]]
         <NDArray 1x2 @cpu(0)>
```