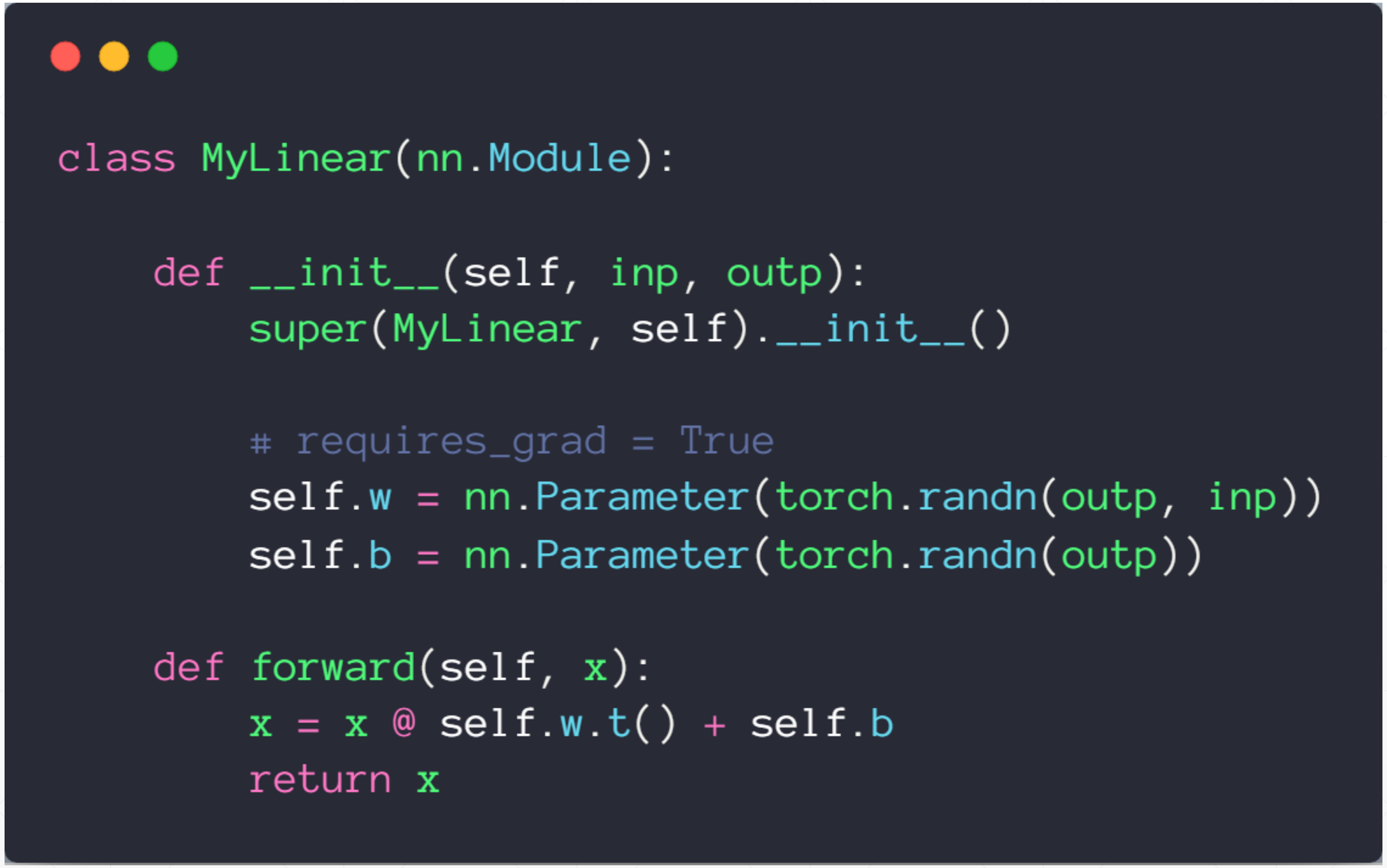
PyTorch

nn.Module

主讲人：龙良曲



```python
class MyLinear(nn.Module):

    def __init__(self, inp, outp):
        super(MyLinear, self).__init__()

        # requires_grad = True
        self.w = nn.Parameter(torch.randn(outp, inp))
        self.b = nn.Parameter(torch.randn(outp))

    def forward(self, x):
        x = x @ self.w.t() + self.b
        return x
```

Magic

- Every Layer is nn.Module
  - nn.Linear
  - nn.BatchNorm2d
  - nn.Conv2d

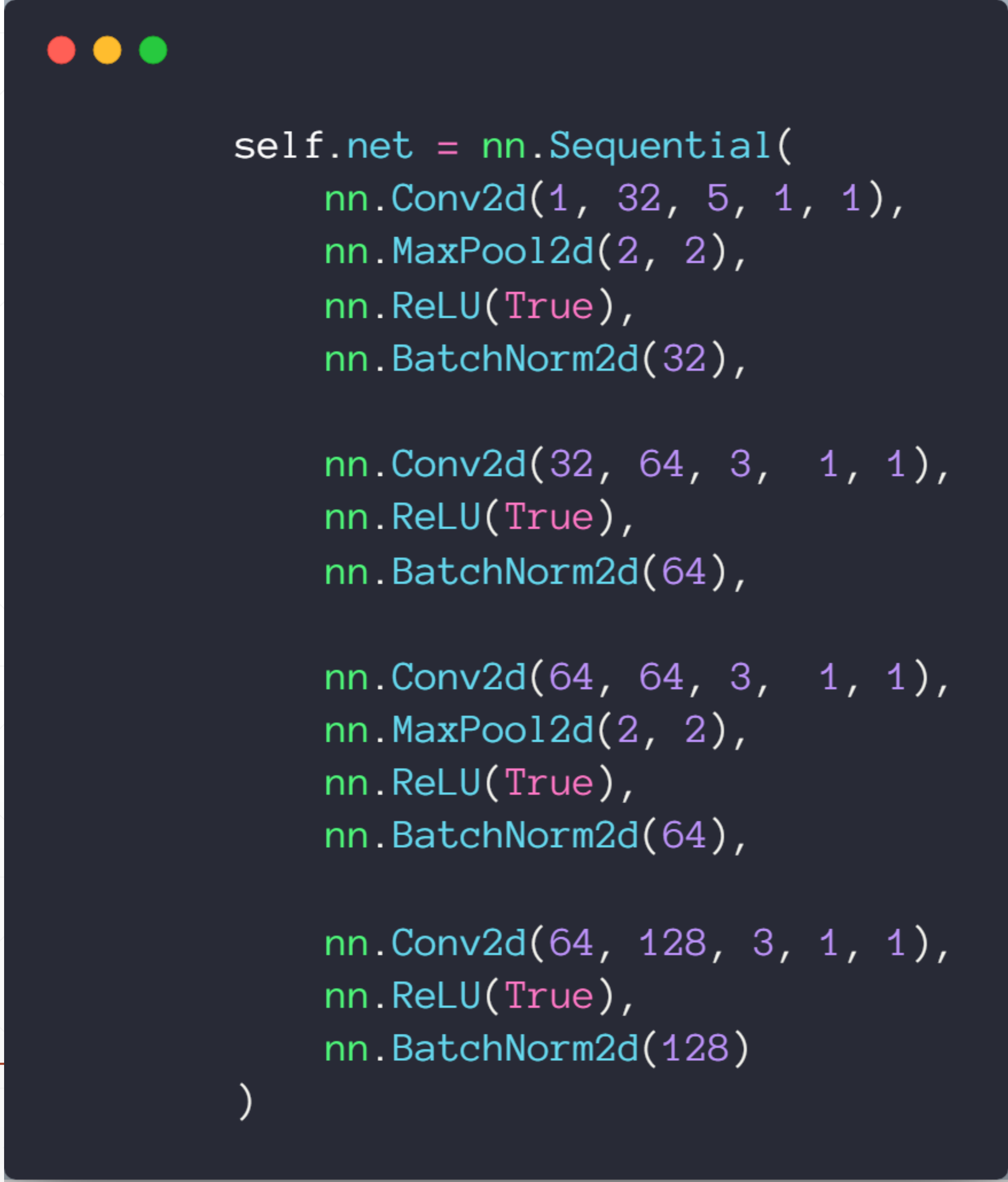
- nn.Module nested in nn.Module

1. embed current layers

- Linear
- ReLU
- Sigmoid
- Conv2d
- ConvTransposed2d
- Dropout
- etc.

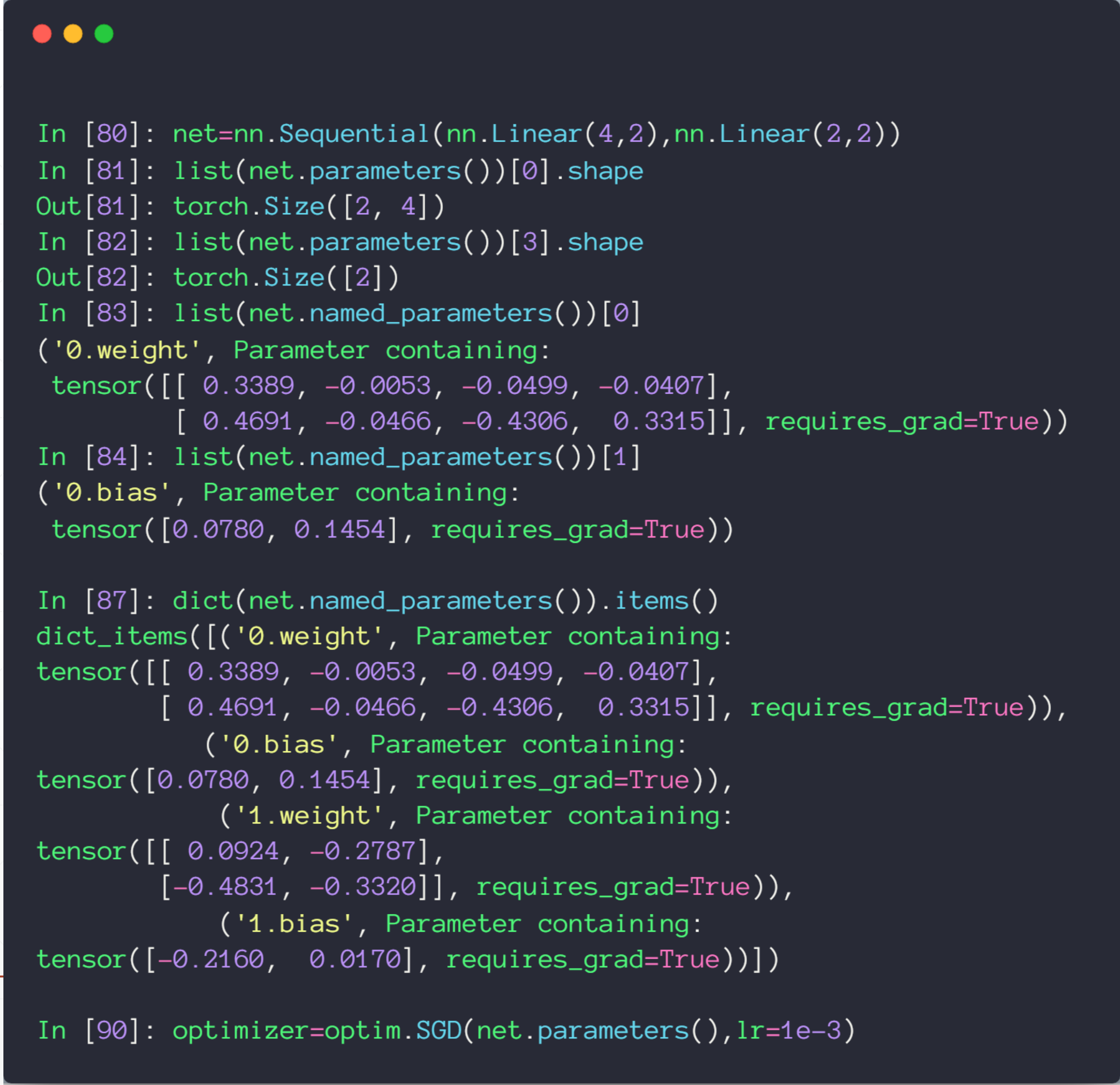
2. Container

- net(x)



```
self.net = nn.Sequential(
    nn.Conv2d(1, 32, 5, 1, 1),
    nn.MaxPool2d(2, 2),
    nn.ReLU(True),
    nn.BatchNorm2d(32),

    nn.Conv2d(32, 64, 3,  1, 1),
    nn.ReLU(True),
    nn.BatchNorm2d(64),

    nn.Conv2d(64, 64, 3,  1, 1),
    nn.MaxPool2d(2, 2),
    nn.ReLU(True),
    nn.BatchNorm2d(64),

    nn.Conv2d(64, 128, 3, 1, 1),
    nn.ReLU(True),
    nn.BatchNorm2d(128)
)
```

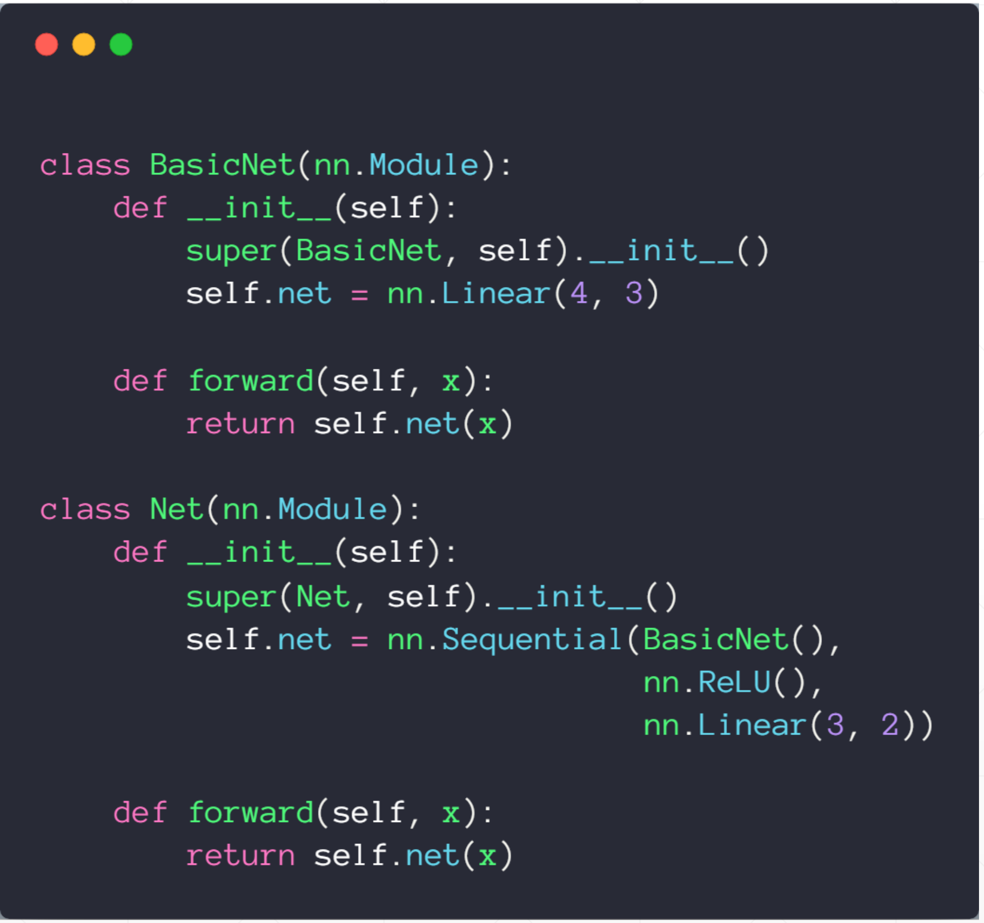
3. parameters



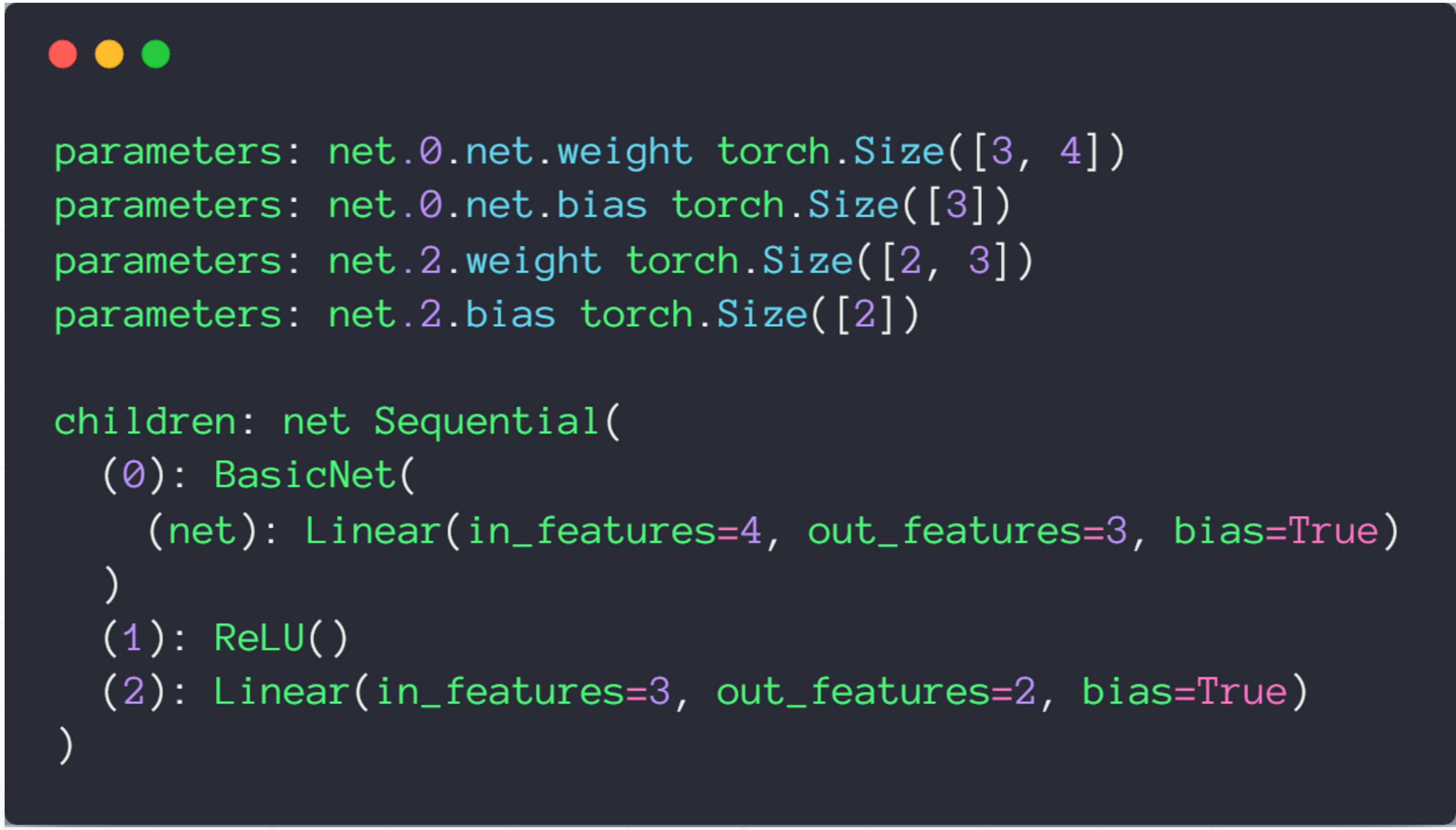
```
In [80]: net=nn.Sequential(nn.Linear(4,2),nn.Linear(2,2))
In [81]: list(net.parameters())[0].shape
Out[81]: torch.Size([2, 4])
In [82]: list(net.parameters())[3].shape
Out[82]: torch.Size([2])
In [83]: list(net.named_parameters())[0]
('0.weight', Parameter containing:
 tensor([[ 0.3389, -0.0053, -0.0499, -0.0407],
         [ 0.4691, -0.0466, -0.4306,  0.3315]], requires_grad=True))
In [84]: list(net.named_parameters())[1]
('0.bias', Parameter containing:
 tensor([0.0780, 0.1454], requires_grad=True))

In [87]: dict(net.named_parameters()).items()
dict_items([('0.weight', Parameter containing:
tensor([[ 0.3389, -0.0053, -0.0499, -0.0407],
        [ 0.4691, -0.0466, -0.4306,  0.3315]], requires_grad=True)),
           ('0.bias', Parameter containing:
tensor([0.0780, 0.1454], requires_grad=True)),
            ('1.weight', Parameter containing:
tensor([[ 0.0924, -0.2787],
        [-0.4831, -0.3320]], requires_grad=True)),
            ('1.bias', Parameter containing:
tensor([-0.2160,  0.0170], requires_grad=True))])

In [90]: optimizer=optim.SGD(net.parameters(),lr=1e-3)
```

4. modules

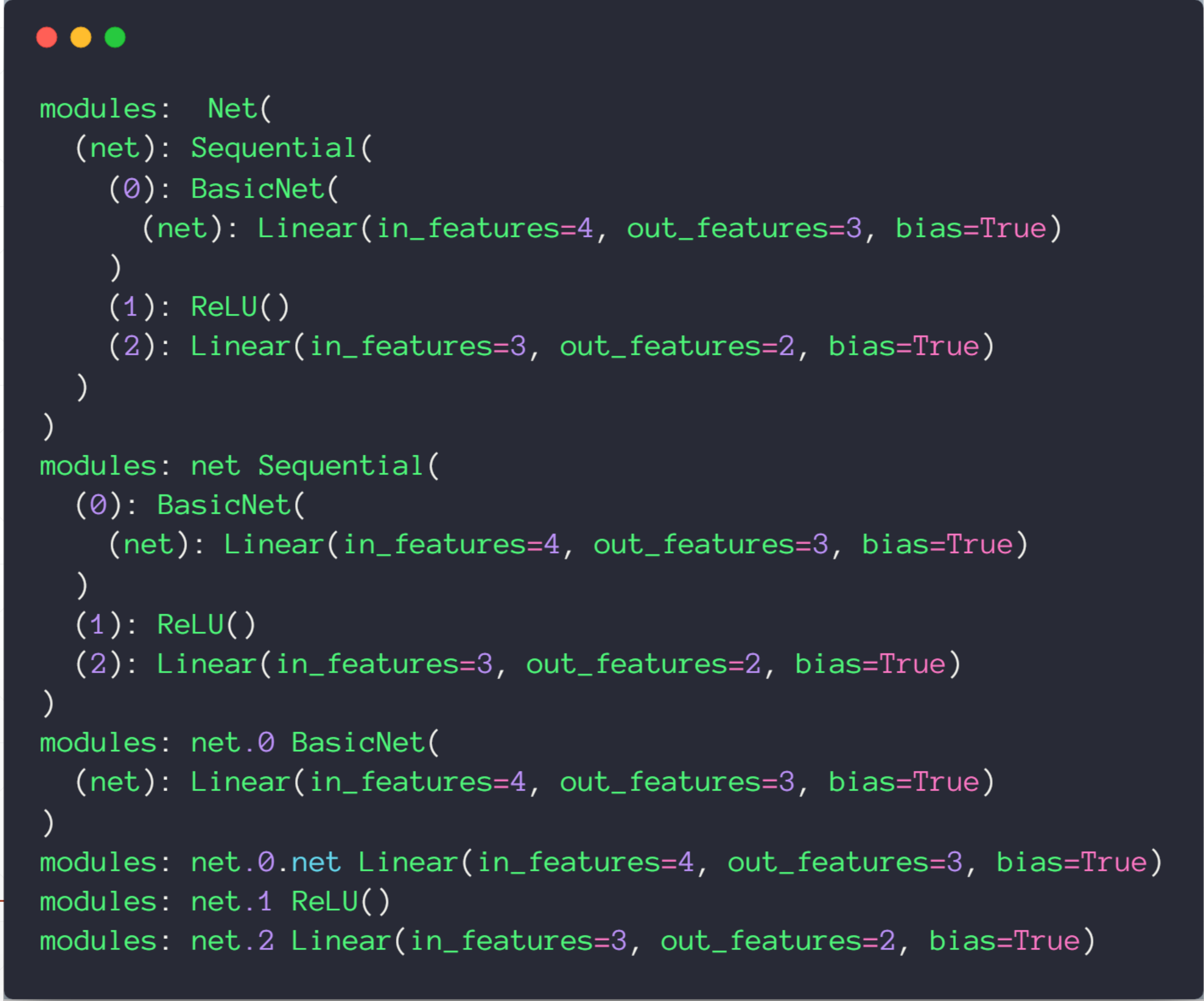
- modules: all nodes
- children: direct children



```python
class BasicNet(nn.Module):
    def __init__(self):
        super(BasicNet, self).__init__()
        self.net = nn.Linear(4, 3)

    def forward(self, x):
        return self.net(x)

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.net = nn.Sequential(BasicNet(),
                                 nn.ReLU(),
                                 nn.Linear(3, 2))

    def forward(self, x):
        return self.net(x)
```

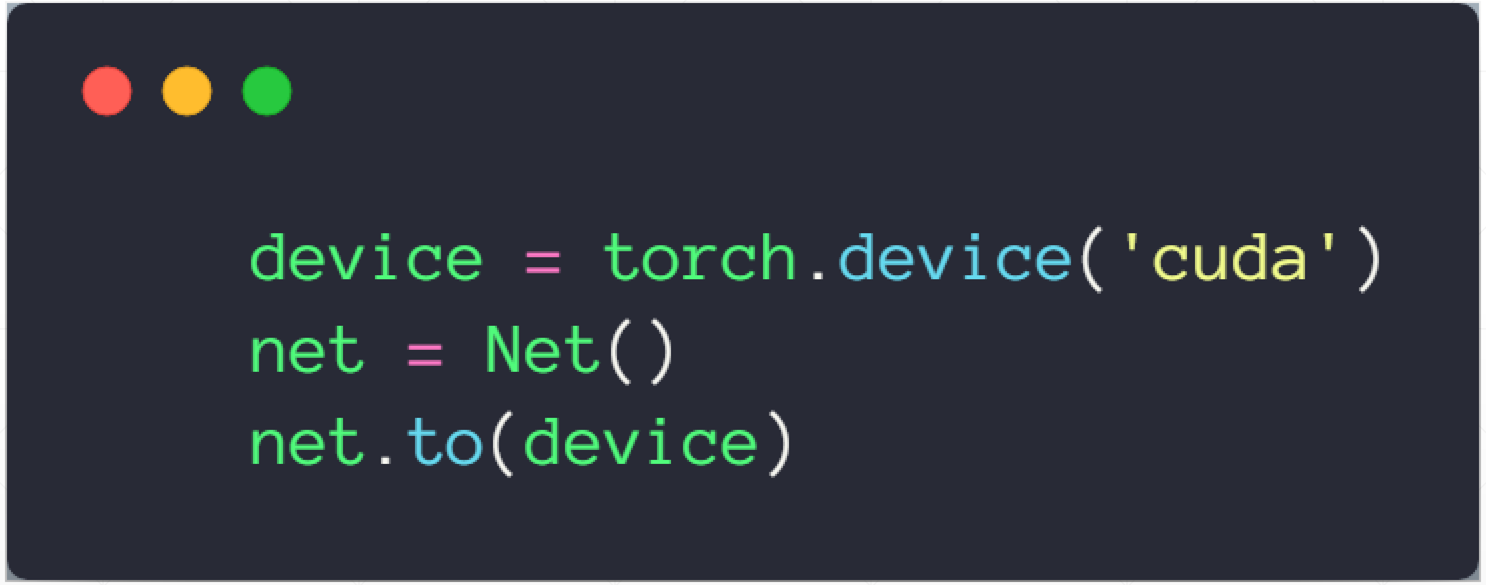


```
parameters: net.0.net.weight torch.Size([3, 4])
parameters: net.0.net.bias torch.Size([3])
parameters: net.2.weight torch.Size([2, 3])
parameters: net.2.bias torch.Size([2])

children: net Sequential(
  (0): BasicNet(
    (net): Linear(in_features=4, out_features=3, bias=True)
  )
  (1): ReLU()
  (2): Linear(in_features=3, out_features=2, bias=True)
)
```



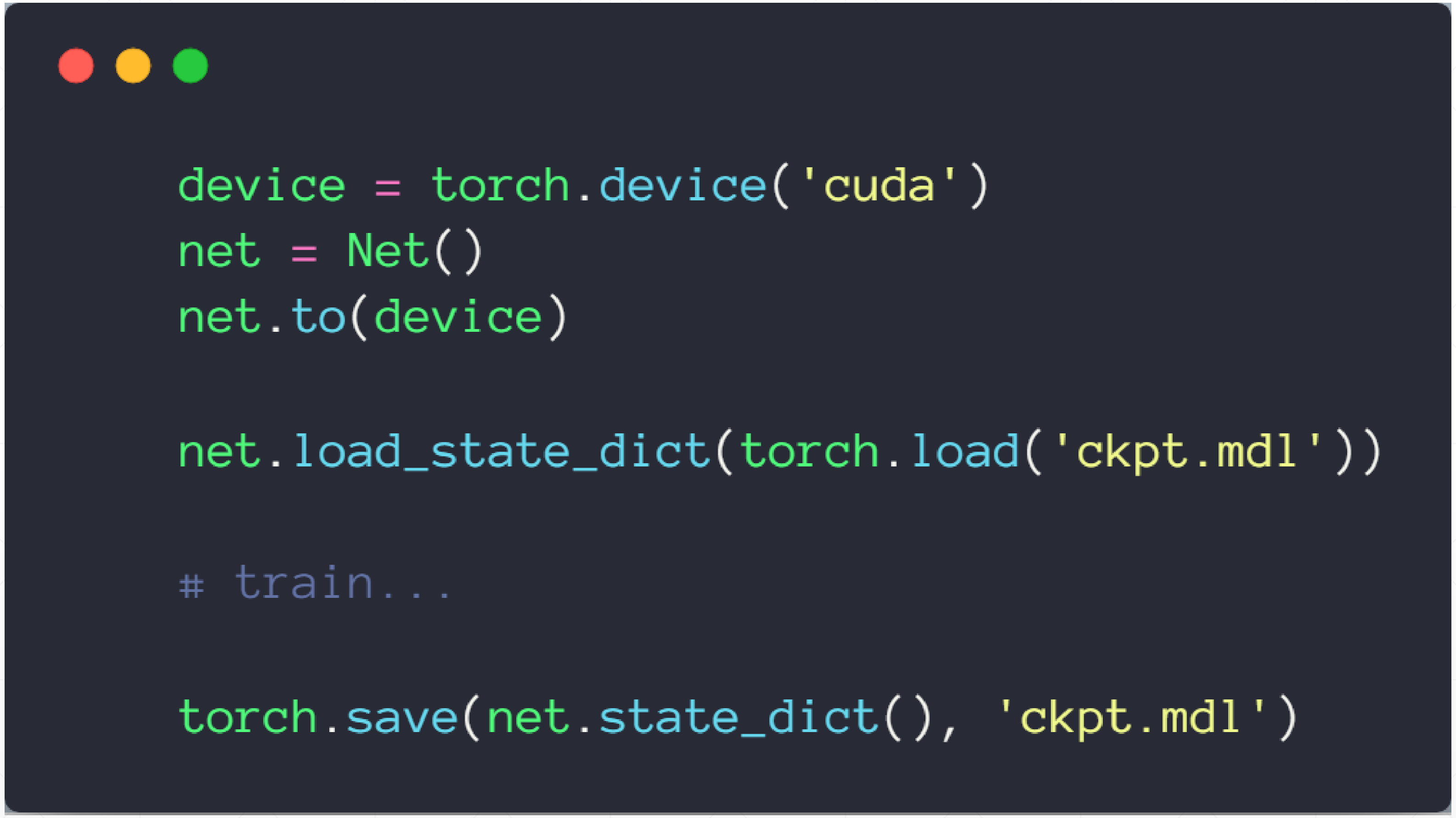
```
modules:  Net(
  (net): Sequential(
    (0): BasicNet(
      (net): Linear(in_features=4, out_features=3, bias=True)
    )
    (1): ReLU()
    (2): Linear(in_features=3, out_features=2, bias=True)
  )
)
modules: net Sequential(
  (0): BasicNet(
    (net): Linear(in_features=4, out_features=3, bias=True)
  )
  (1): ReLU()
  (2): Linear(in_features=3, out_features=2, bias=True)
)
modules: net.0 BasicNet(
  (net): Linear(in_features=4, out_features=3, bias=True)
)
modules: net.0.net Linear(in_features=4, out_features=3, bias=True)
modules: net.1 ReLU()
modules: net.2 Linear(in_features=3, out_features=2, bias=True)
```

5. to(device)



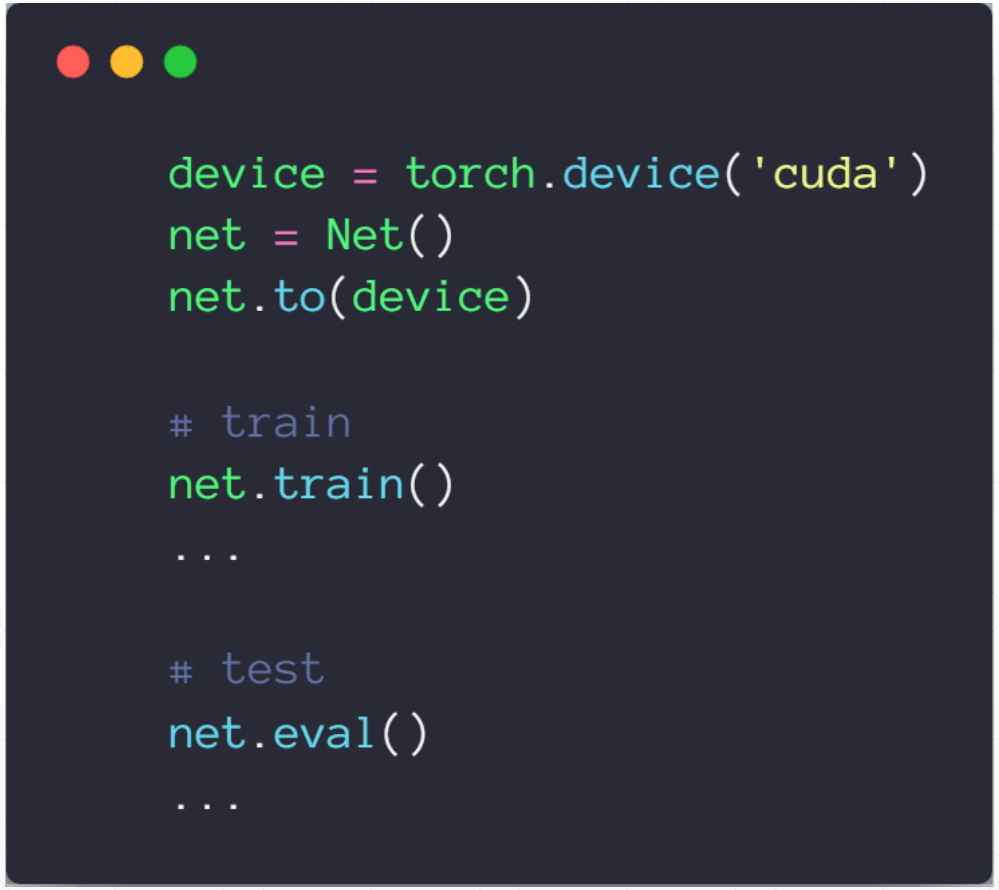
```
device = torch.device('cuda')
net = Net()
net.to(device)
```

6. save and load



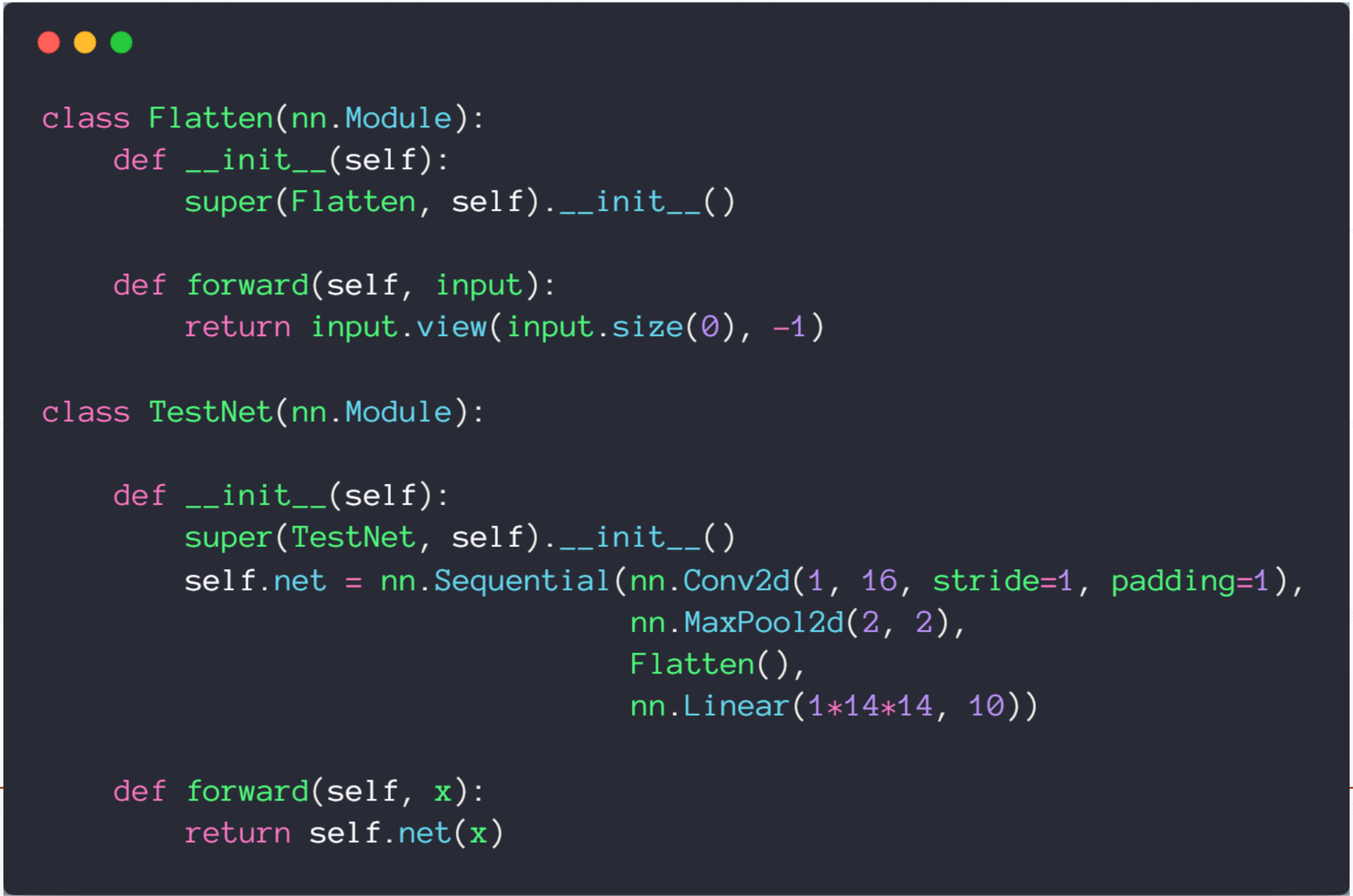
```
device = torch.device('cuda')
net = Net()
net.to(device)

net.load_state_dict(torch.load('ckpt.mdl'))

# train...

torch.save(net.state_dict(), 'ckpt.mdl')
```

7. train/test



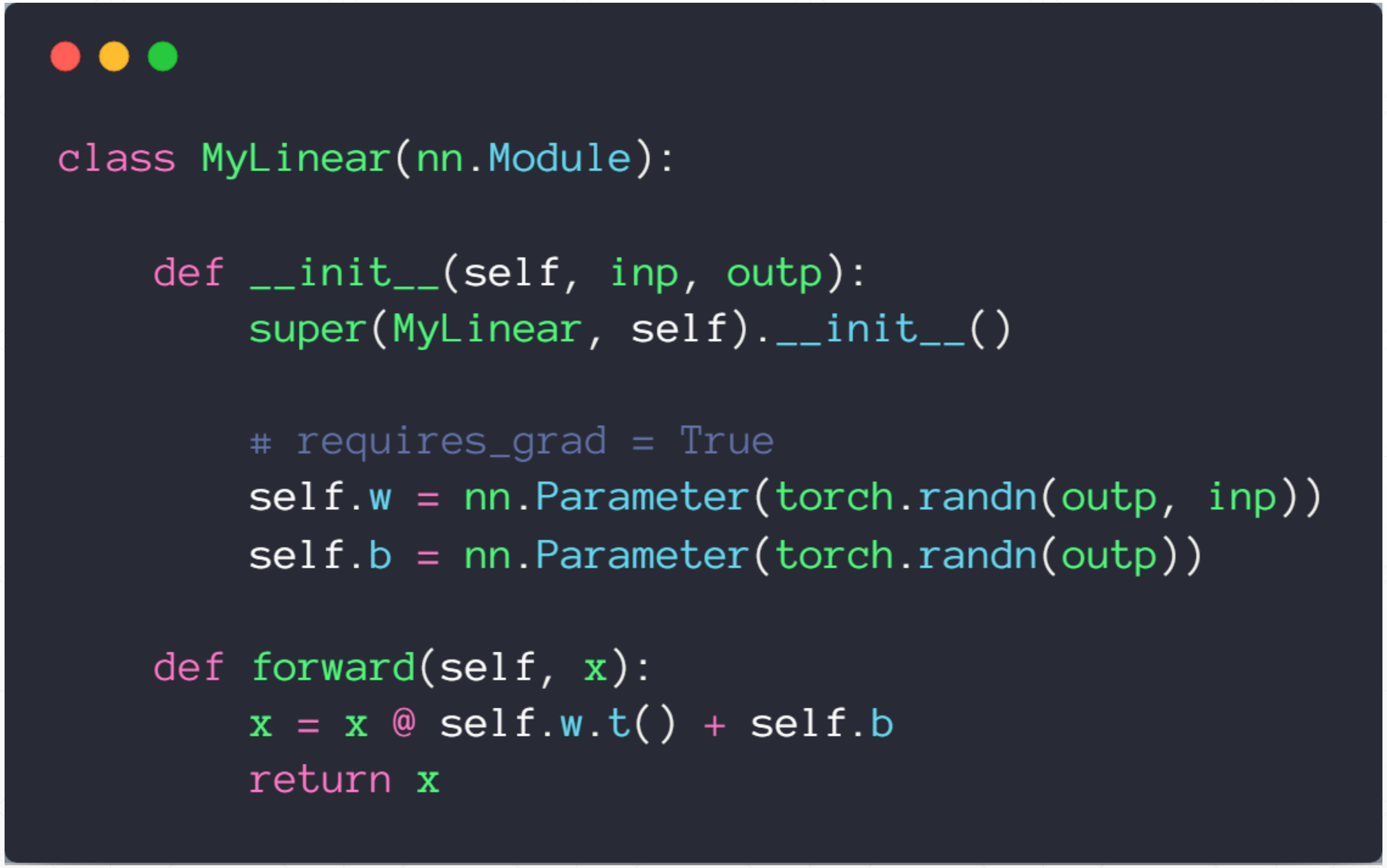
```
device = torch.device('cuda')
net = Net()
net.to(device)

# train
net.train()
...

# test
net.eval()
...
```

8. implement own layer



```
class Flatten(nn.Module):
    def __init__(self):
        super(Flatten, self).__init__()

    def forward(self, input):
        return input.view(input.size(0), -1)

class TestNet(nn.Module):

    def __init__(self):
        super(TestNet, self).__init__()
        self.net = nn.Sequential(nn.Conv2d(1, 16, stride=1, padding=1),
                                 nn.MaxPool2d(2, 2),
                                 Flatten(),
                                 nn.Linear(1*14*14, 10))


    def forward(self, x):
        return self.net(x)
```

8. own linear layer



```python
class MyLinear(nn.Module):

    def __init__(self, inp, outp):
        super(MyLinear, self).__init__()

        # requires_grad = True
        self.w = nn.Parameter(torch.randn(outp, inp))
        self.b = nn.Parameter(torch.randn(outp))

    def forward(self, x):
        x = x @ self.w.t() + self.b
        return x
```

下一课时

Data Argumentation

Thank You.