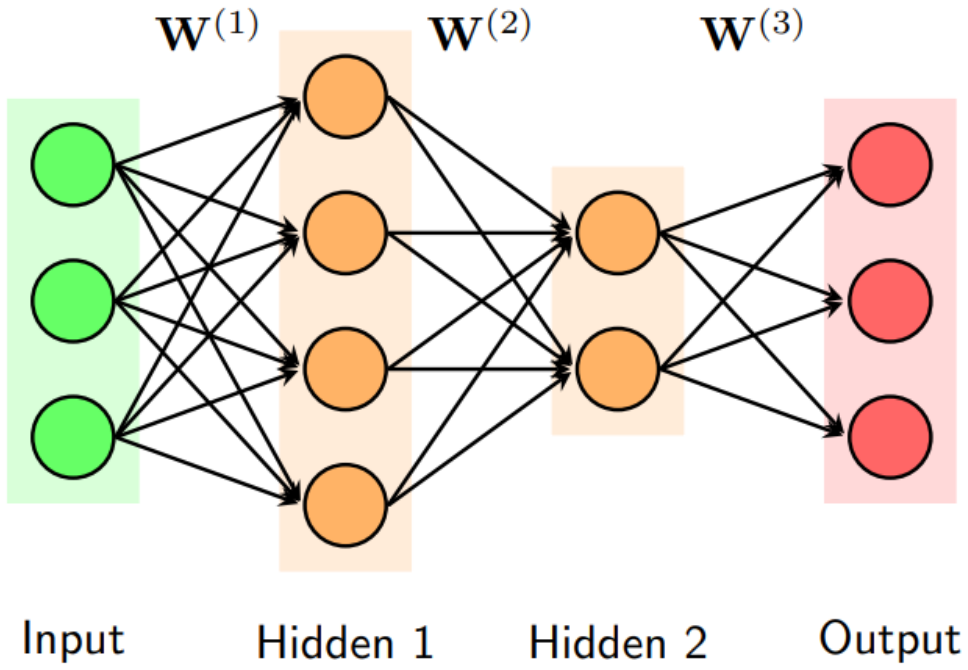
CSE: **Faculty of Computer Science and Engineering**
Thuyloi University

Tính toán trong mạng học sâu

TS. Nguyễn Thị Kim Ngân

Tầng (Layer)

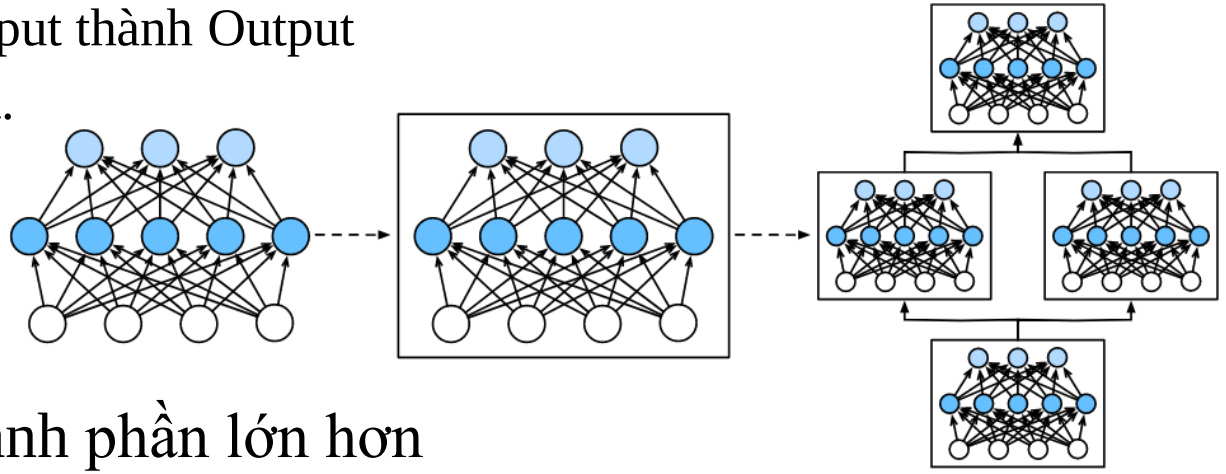
- Lớp (layer) là tập hợp chứa các neuron theo chiều dọc
- Mỗi neural network có 3 loại layer chính:
  - Lớp đầu vào (input layer)
  - Lớp ẩn (hidden layer)
  - Lớp đầu ra (output layer)



- Số lượng layer bằng số hidden layers cộng với 1
- Số lượng layer trong một MLP được ký hiệu là L
  - Trong hình bên, L=3

Khối (Block)

- Một khối có thể là một tầng duy nhất, một mảng đa tầng hoặc toàn bộ mô hình
- Một khối:
  - Input
  - Output
  - Các tham số cần thiết (có một số khối không yêu cầu bất kỳ tham số nào)
  - Phương thức forward để chuyển hóa Input thành Output
  - Phương thức backward để tính gradient.



- Có thể kết hợp các khối để tạo ra thành phần lớn hơn

Ví dụ

- Tạo ra một mạng gồm:
  - Một tầng ẩn kết nối đầy đủ, có 256 nút, sử dụng hàm kích hoạt ReLUt
  - Tầng đầu ra kết nối đầy đủ với 10 nút (không có hàm kích hoạt)
- Dùng Dense để khai báo từng layer
- Dùng Sequential để khai báo từng block

```
from mxnet import np, npx
from mxnet.gluon import nn
npx.set_np()

x = np.random.uniform(size=(2, 20))

net = nn.Sequential()
net.add(nn.Dense(256, activation='relu'))
net.add(nn.Dense(10))
net.initialize()
net(x)
```

Một khối tùy chỉnh

```
From mxnet.gluon import nn

#Definition of a block class

class MLP(nn.Block):

    # Declare a layer with model parameters. Here, we declare
    two fully connected layers

    def __init__(self, **kwargs):

        super(MLP, self).__init__(**kwargs)

        #Hidden layer

        self.hidden = nn.Dense(256, activation='relu’)

        # Output layer

        self.output = nn.Dense(10)


    def forward(self, x):

        return self.output(self.hidden(x))
```

```
#Using a instance block

net = MLP()

net.initialize()

net(x)
```

Khối tuần tự

```
From mxnet.gluon import nn
class MySequential(nn.Block):
    def add(self, block):
        self._children[block.name] = block

    def forward(self, x):
        for block in self._children.values():
            x = block(x)
        return x
```

```
net = MySequential()
net.add(nn.Dense(256, activation='relu’))
 net.add(nn.Dense(10))
net.initialize()
net(x)
```

Lồng các khối

```
from mxnet.gluon import nn

class NestMLP(nn.Block):

    def __init__(self, **kwargs):
        super(NestMLP, self).__init__(**kwargs)
        self.net = nn.Sequential()
        self.net.add(nn.Dense(64, activation='relu'),
                     nn.Dense(32, activation='relu'))
        self.dense = nn.Dense(16, activation='relu')


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

```
chimera = nn.Sequential()

chimera.add(NestMLP(), nn.Dense(20), MLP())

chimera.initialize()

chimera(x)
```

Quản lý tham số

- Ví dụ mạng Perceptron có một tầng ẩn

```
from mxnet import init, np, npx
from mxnet.gluon import nn
npx.set_np()
net = nn.Sequential()
net.add(nn.Dense(256, activation='relu’))
net.add(nn.Dense(10))
net.initialize() # Use the default initialization method
x = np.random.uniform(size=(2, 20))
net(x) # Forward computation
```

```
#Truy cập tham số theo từng tầng
net[0].params
net[1].params

#Truy cập tham số qua thuộc tính bias/weight
net[1].bias
net[1].bias.data()

#Truy cập các tham số theo tên của chúng
net[0].params['dense0_weight
net[0].params['dense0_weight'].data()

#Truy cập đạo hàm

net[0].weight.grad()
```

Quản lý tham số

- Ví dụ mạng Perceptron có một tầng ẩn

```
from mxnet import init, np, npx
from mxnet.gluon import nn
npx.set_np()
net = nn.Sequential()
net.add(nn.Dense(256, activation='relu’))
net.add(nn.Dense(10))
net.initialize() # Use the default initialization method
x = np.random.uniform(size=(2, 20))
net(x) # Forward computation
```

```
#Truy cập tất cả các tham số
# Tham số của tầng đầu tiên
net[0].collect_params()
# Tham số của toàn bộ mạng
 net.collect_params()
# Truy cập các tham số của mạng:
net.collect_params()['dense1_bias'].data()

#Truy cập theo tên của tham số
net.collect_params('.*weight')
net.collect_params('dense0.*')
```

Thu thập tham số từ các khối lồng nhau

```
def block1():
    net = nn.Sequential()
    net.add(nn.Dense(32, activation='relu'))
    net.add(nn.Dense(16, activation='relu'))
    return net

def block2():
    net = nn.Sequential()
    for i in range(4):
    net.add(block1())
    return net
```

```
rgnet = nn.Sequential()
rgnet.add(block2())
rgnet.add(nn.Dense(10))
rgnet.initialize()
rgnet(x)

#Thu thập tất cả tham số
rgnet.collect_params
rgnet.collect_params()

# Truy cập tham số cụ thể
rgnet[0][1][0].bias.data()
```

Khởi tạo tham số (có sẵn)

- Khởi tạo tất cả các tham số với các biến ngẫu nhiên Gauss có độ lệch chuẩn bằng 0.01.

```
net.initialize(init=init.Normal(sigma=0.01), force_reinit=True)
net[0].weight.data()[0]
```

- Khởi tạo tất cả tham số với một hằng số, bằng cách sử dụng bộ khởi tạo Constant(1)

```
net.initialize(init=init.Constant(1), force_reinit=True) net[0].weight.data()[0]
```

- Khởi tạo tham số cho các khối khác nhau

```
net[0].weight.initialize(init=init.Xavier(), force_reinit=True)
net[1].initialize(init=init.Constant(42), force_reinit=True)
net[1].weight.data()[0, 0]
```

Khởi tạo tham số (tùy chỉnh)

```
class MyInit(init.Initializer):
    def _init_weight(self, name, data):
        print('Init', name, data.shape)
        data[:] = np.random.uniform(-10, 10, data.shape)
        data *= np.abs(data) >= 5

net.initialize(MyInit(), force_reinit=True) net[0].weight.data()[0]
```

Khởi tạo tham số (tùy chỉnh)

```
net[0].weight.data()[:] += 1
net[0].weight.data()[0, 0] = 42
net[0].weight.data()[0]
```