Python Learning

5.4 : Implementation of simple layer
5.5 : Implementation of activation function layer

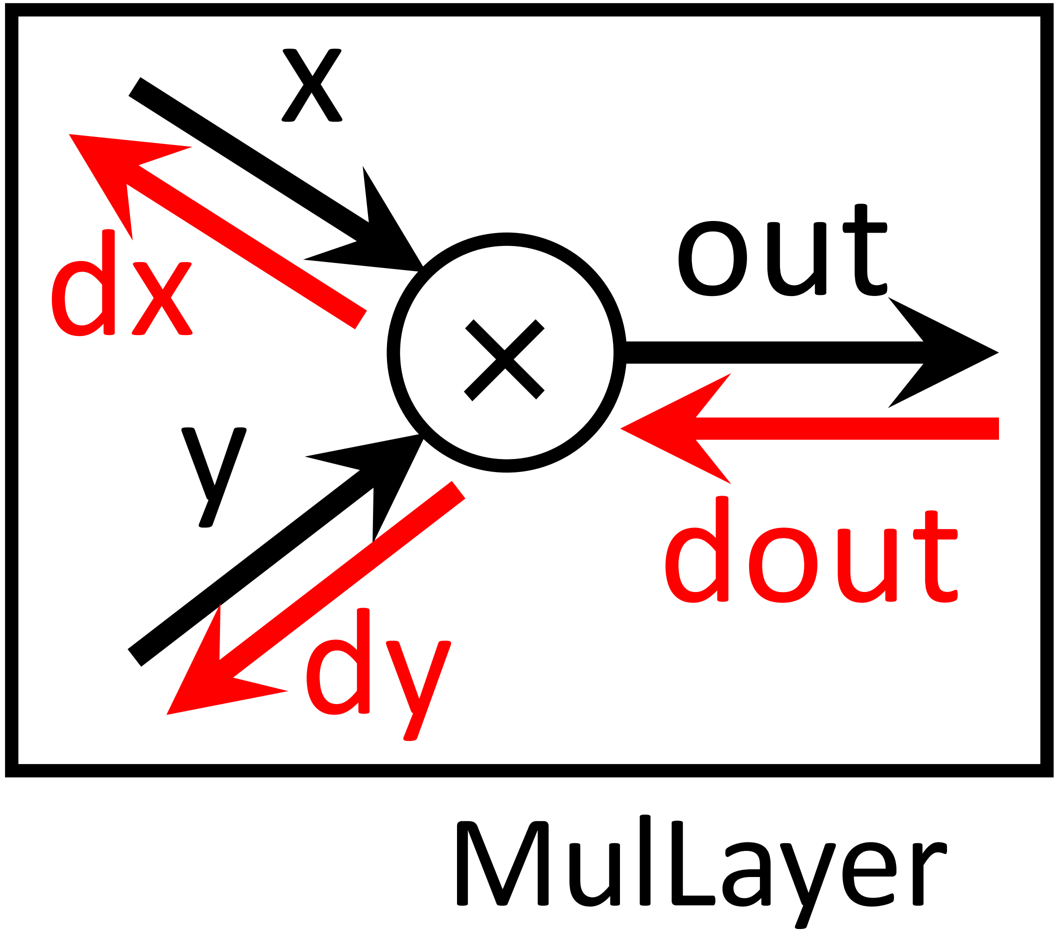
M2 Hayato Tsuda

Overview of this presentation

- 5.4 Implementation of simple layer
  - Multiplication layer
  - Addtion layer
- 5.5 Implementation of activation function layer
  - ReLU layer
  - Sigmoid layer

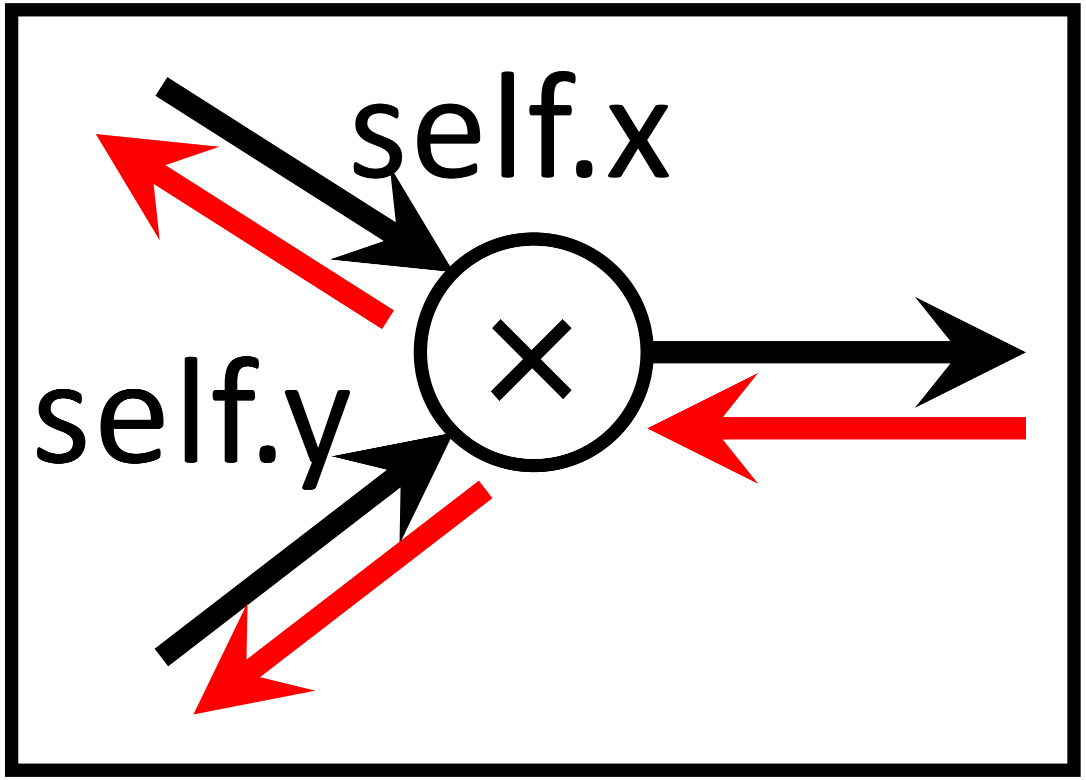
Implementation of multiplication layer

```
class MulLayer:
    def __init__(self):
        self.x = None
        self.y = None

    def forward(self, x, y):
        self.x = x
        self.y = y
        out = x * y

        return out

    def backward(self, dout):
        dx = dout * self.y
        dy = dout * self.x

        return dx, dy
```



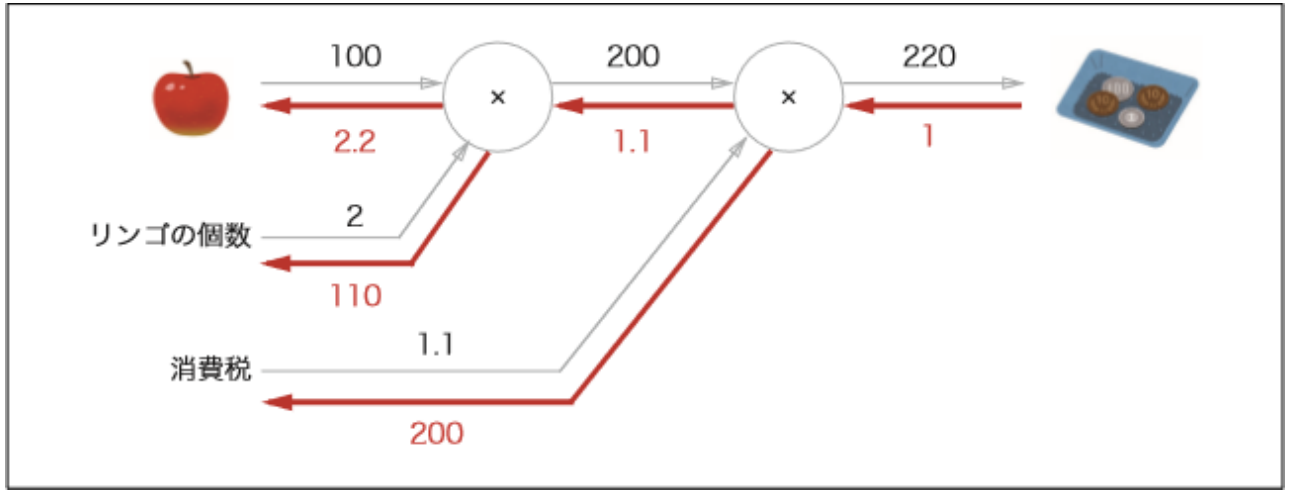
Implementation of multiplication layer

```
class MulLayer:
    def __init__(self):
        self.x = None
        self.y = None

    def forward(self, x, y):
        self.x = x
        self.y = y
        out = x * y

        return out

    def backward(self, dout):
        dx = dout * self.y
        dy = dout * self.x

        return dx, dy
```



MulLayer

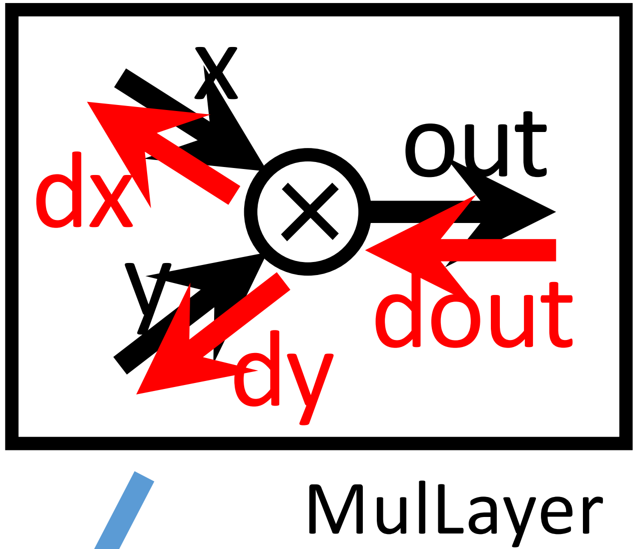
Implementation of “Buy Apples”

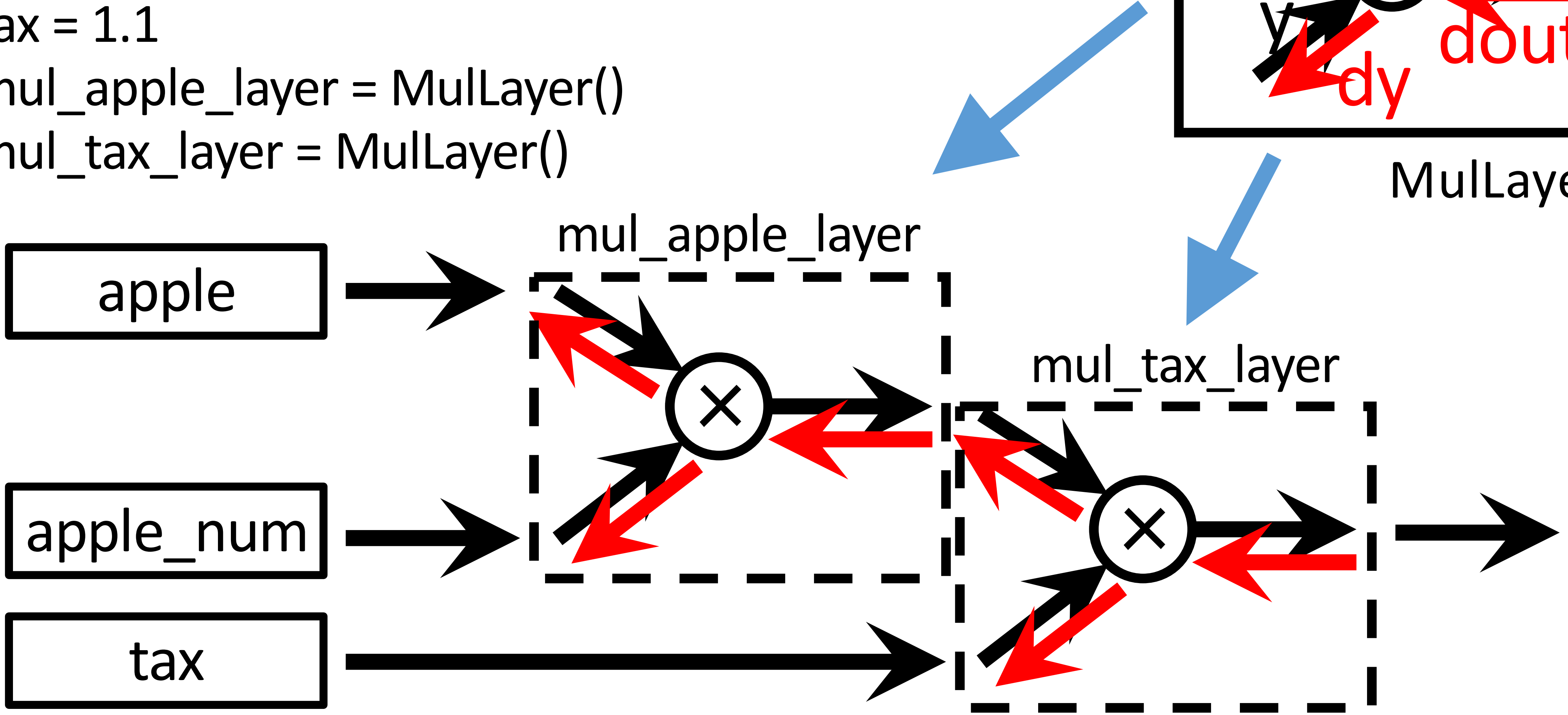



```
apple = 100
apple_num = 2
tax = 1.1

mul_apple_layer = MulLayer()
mul_tax_layer = MulLayer()

# forward
apple_price = mul_apple_layer.forward(apple, apple_num)
price = mul_tax_layer.forward(apple_price, tax)

print(price) # 220

# backward
dprice = 1
dapple_price, dtax = mul_tax_layer.backward(dprice)
dapple, dapple_num = mul_apple_layer.backward(dapple_price)

print(dapple, dapple_num, dtax) # 2.2 110 200
```

Implementation of “Buy Apples”

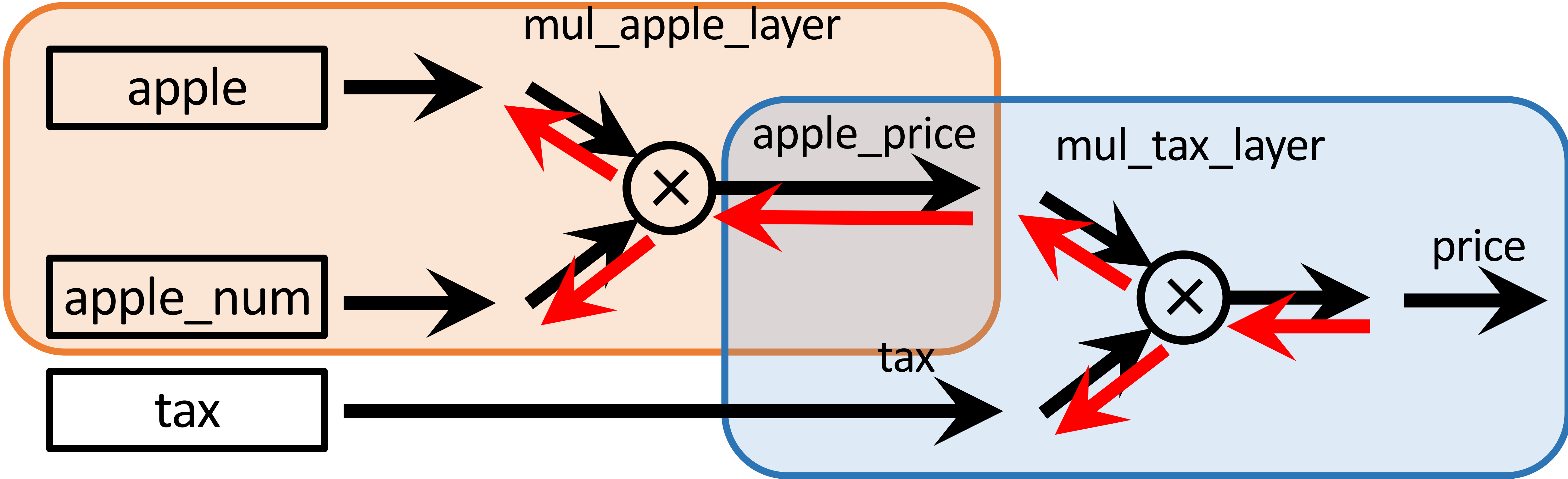
```
apple = 100
apple_num = 2
tax = 1.1
mul_apple_layer = MulLayer()
mul_tax_layer = MulLayer()
```





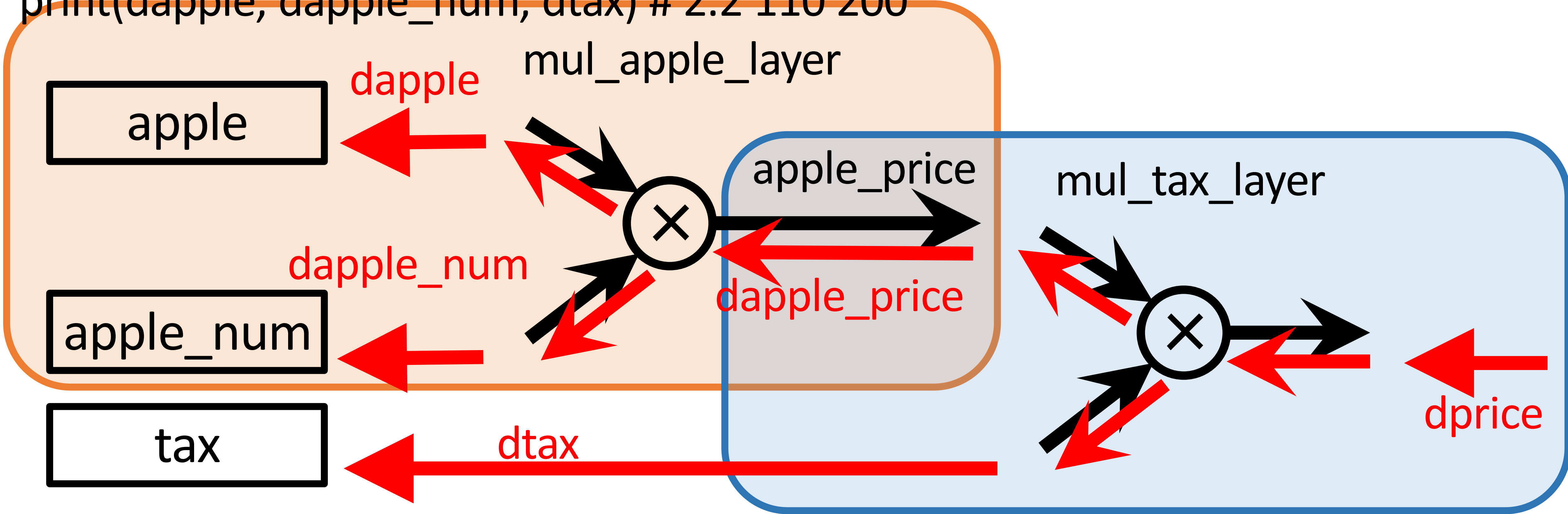
Implementation of “Buy Apples”

```
# forward
apple_price = mul_apple_layer.forward(apple, apple_num)
price = mul_tax_layer.forward(apple_price, tax)
print(price) # 220
```



Implementation of “Buy Apples”

```
# backward
dprice = 1
dapple_price, dtax = mul_tax_layer.backward(dprice)
dapple, dapple_num = mul_apple_layer.backward(dapple_price)
print(dapple, dapple_num, dtax) # 2.2 110 200
```



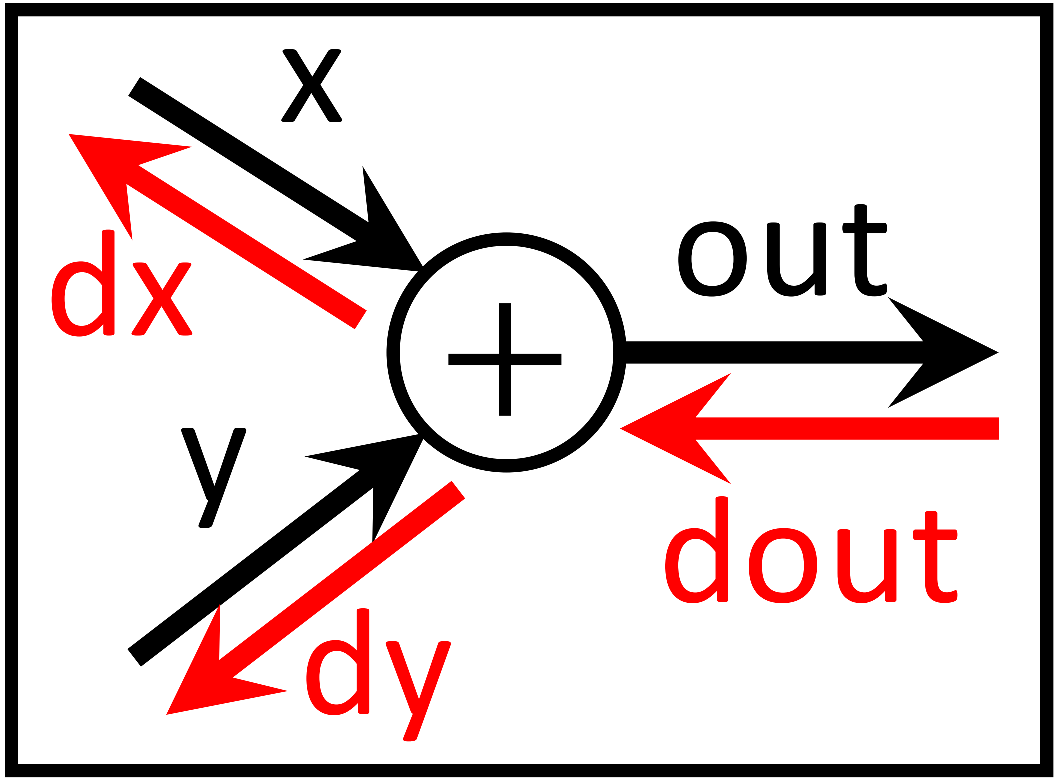
Implementation of “Buy Apples”

```
print(price) # 220
print(dapple, dapple_num, dtax) # 2.2 110 200
```

```
MacBook-Pro:PythonLearning tsuda$ python3 BuyApple.py
220.00000000000003
2.2 110.00000000000001 200
```

Implementation of addition layer

```
class AddLayer:
    def __init__(self):
        pass

    def forward(self, x, y):
        out = x + y
        return out

    def backward(self, dout):
        dx = dout * 1
        dy = dout * 1
        return dx, dy
```



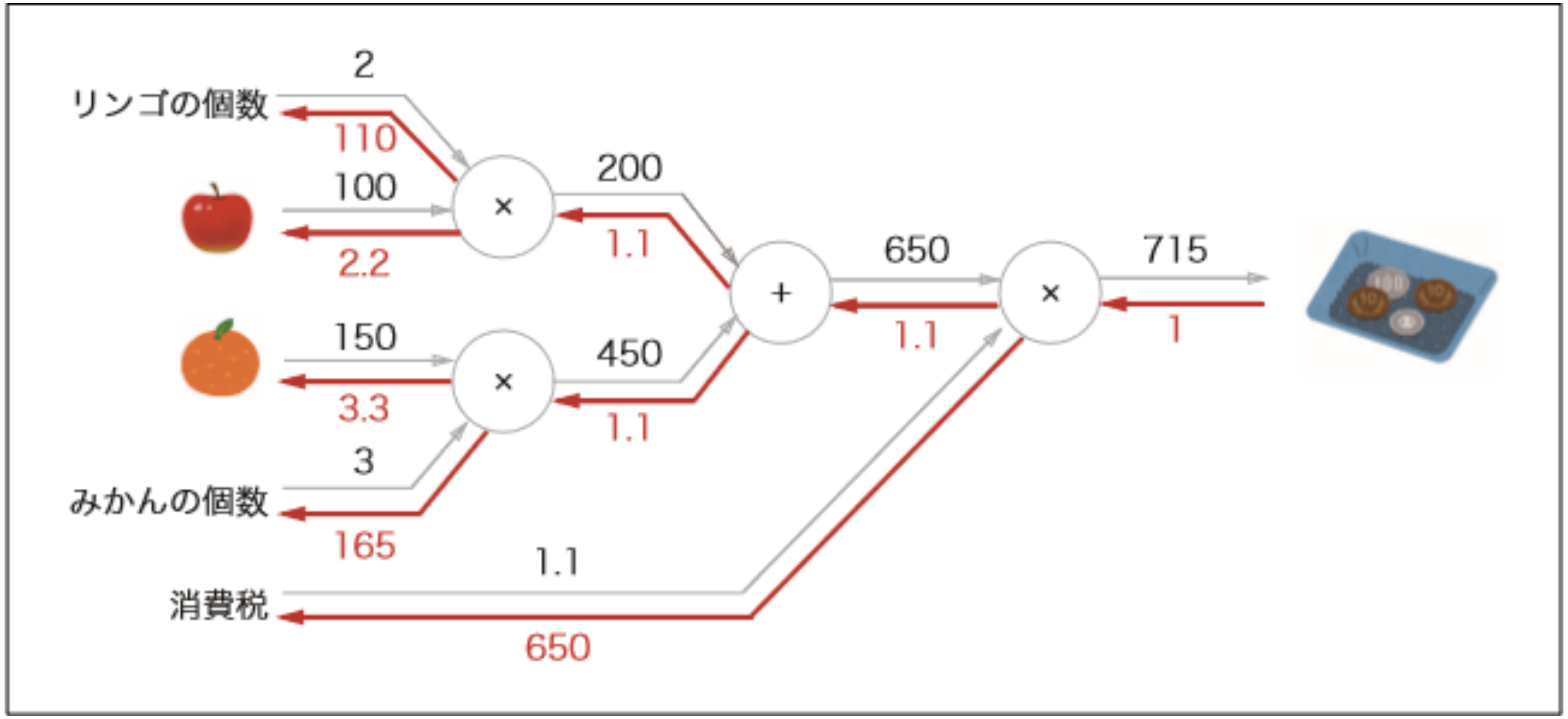
MulLayer

Implementation of “Buy apples and oranges”

BuyFruits.py

```python

# Buy apple and orange
apple = 100
apple_num = 2
orange = 150
orange_num = 3
tax = 1.1

# layer
mul_apple_layer = MulLayer()
mul_orange_layer = MulLayer()
add_apple_orange_layer = AddLayer()
mul_tax_layer = MulLayer()

# forward
apple_price = mul_apple_layer.forward(apple, apple_num) #(1)
orange_price = mul_orange_layer.forward(orange, orange_num) #(2)
all_price = add_apple_orange_layer.forward(apple_price, orange_price) #(3)
price = mul_tax_layer.forward(all_price, tax) #(4)

# backward
dprice = 1
dall_price, dtax = mul_tax_layer.backward(dprice) #(4)
dapple_price, dorange_price = add_apple_orange_layer.backward(dall_price) #(3)
dorange, dorange_num = mul_orange_layer.backward(dorange_price) #(2)
dapple, dapple_num = mul_apple_layer.backward(dapple_price) #(1)

print(price) # 715
print(dapple_num, dapple, dorange, dorange_num, dtax) # 110 2.2 3.3 165 650
```

Implementation of “Buy apples and oranges”

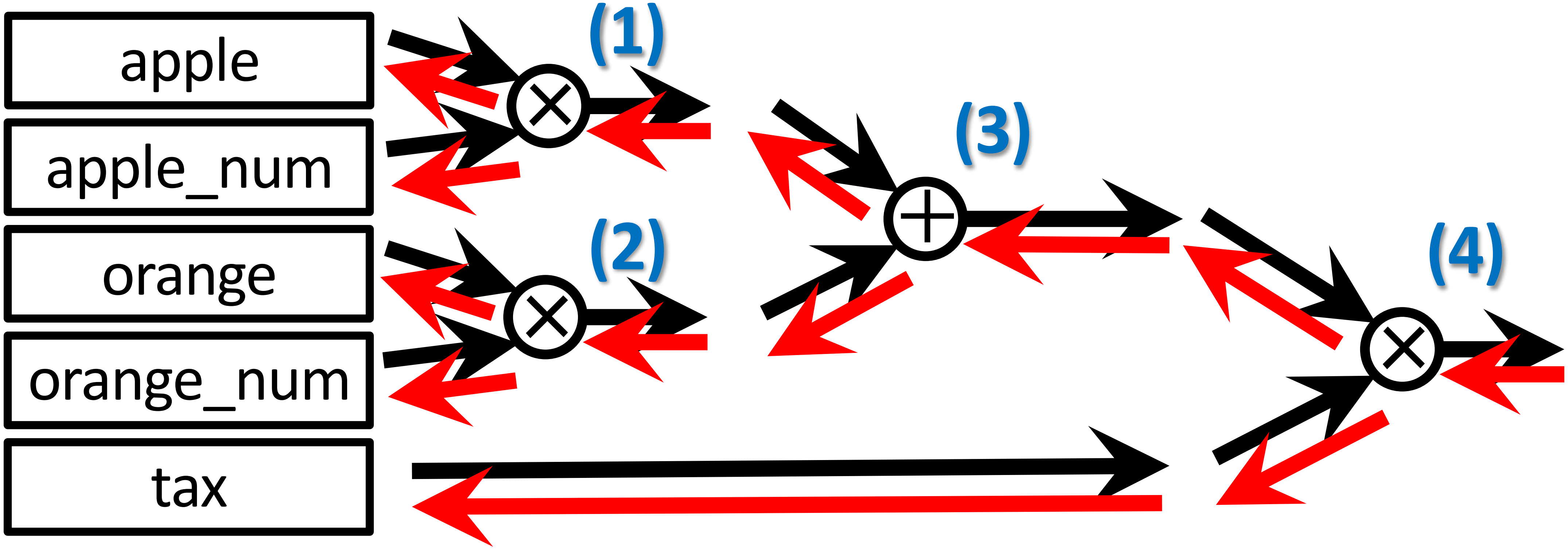


apple = 100
apple_num = 2
orange = 150
orange_num = 3
tax = 1.1

\# layer
mul_apple_layer = MulLayer()
mul_orange_layer = MulLayer()
add_apple_orange_layer = AddLayer()
mul_tax_layer = MulLayer()

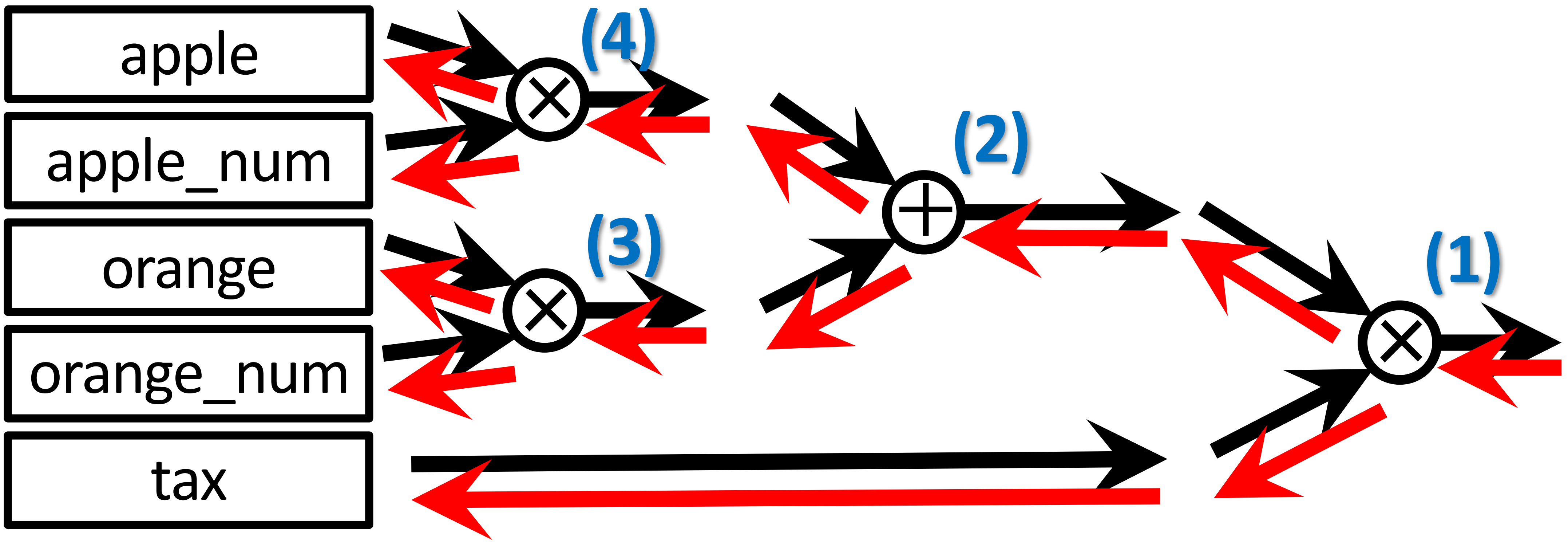
Implementation of “Buy apples and oranges”

```
# forward
apple_price = mul_apple_layer.forward(apple, apple_num) #(1)
orange_price = mul_orange_layer.forward(orange, orange_num) #(2)
all_price = add_apple_orange_layer.forward(apple_price, orange_price) #(3)
price = mul_tax_layer.forward(all_price, tax) #(4)
```



Implementation of "Buy apples and oranges"

```
# backward
dprice = 1
dall_price, dtax = mul_tax_layer.backward(dprice) #(4)
dapple_price, dorange_price = add_apple_orange_layer.backward(dall_price) #(3)
dorange, dorange_num = mul_orange_layer.backward(dorange_price) #(2)
dapple, dapple_num = mul_apple_layer.backward(dapple_price) #(1)
```



Implementation of “Buy apples and oranges”

print(price) # 715
print(dapple_num, dapple, dorange, dorange_num, dtax) # 110 2.2 3.3 165 650

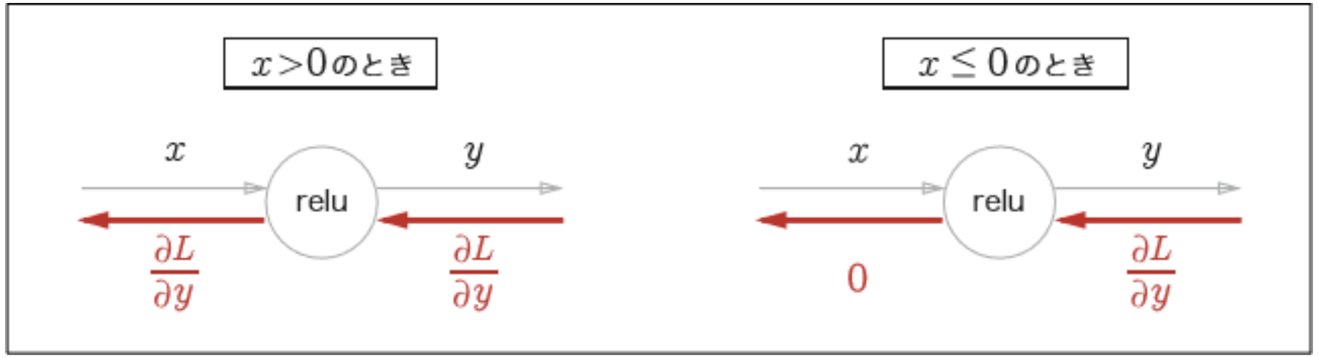
```
MacBook-Pro:PythonLearning tsuda$ python3 BuyFruits.py
715.0000000000001
110.00000000000001 2.2 3.3000000000000003 165.0 650
```

Implementation of activation function layer

- Apply the idea of computational graph to **neural network**
- Implement layers as classes
- At first, implement activation function layer
    - ReLU layer
    - Sigmoid layer

Implementation of ReLU layer

- ReLU function : $$y = \begin{cases} x \ (x > 0) \\ 0 \ (x \leq 0) \end{cases}$$
- A differential of ReLU : $$\frac{\partial y}{\partial x} = \begin{cases} 1 \ (x > 0) \\ 0 \ (x \leq 0) \end{cases}$$
- ReLU behaves like as a switch.



Implementation of ReLU layer

```
class Relu:
    def __init__(self):
        self.mask = None

    def forward(self, x):
        self.mask = (x <= 0)
        out = x.copy()
        out[self.mask] = 0

        return out

    def backward(self, dout):
        dout[self.mask] = 0
        dx = dout

        return dx
```

About “Boolean ndarray”

```
>>> x = np.array( [[1.0, -0.5], [-2.0, 3.0]] )
>>> mask = (x <= 0)
>>> print(x)
[[ 1.  -0.5]
 [-2.   3. ]]
>>> print(mask)
[[False  True]
 [ True False]]
>>>
>>> print(x[mask])
[-0.5 -2. ]
>>> print(x[mask]+1)
[ 0.5 -1. ]
>>> x = x[mask] + 2
>>> print(x)
[1.5 0. ]
```

References : https://hydrocul.github.io/wiki/numpy/ndarray-ref-boolean.html

About "Boolean ndarray"

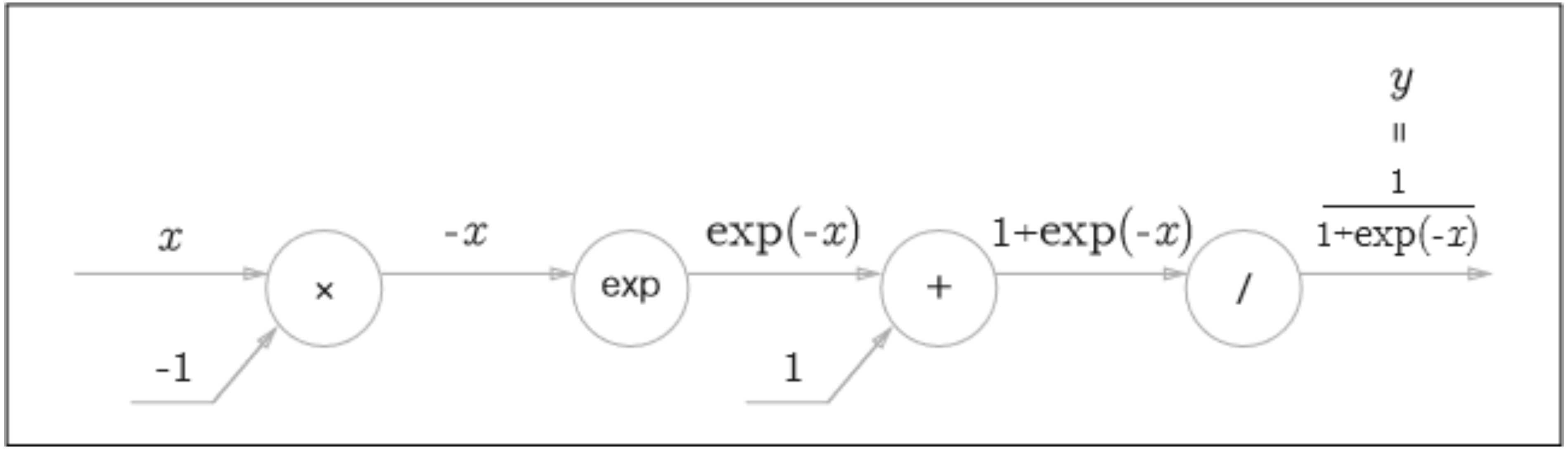
```
>>> x = np.array( [[1.0, -0.5], [-2.0, 3.0]] )
>>> mask = (x <= 0)
>>> print(x)
[[ 1.  -0.5]
 [-2.   3. ]]
>>> print(mask)
[[False  True]
 [ True False]]
>>> x[mask] = 0
>>> print(x)
[[1. 0.]
 [0. 3.]]
>>> x[mask] = [10, 20]
>>> print(x)
[[ 1. 10.]
 [20.  3.]]
```

References : https://hydrocul.github.io/wiki/numpy/ndarray-ref-boolean.html

Implementation of Sigmoid layer

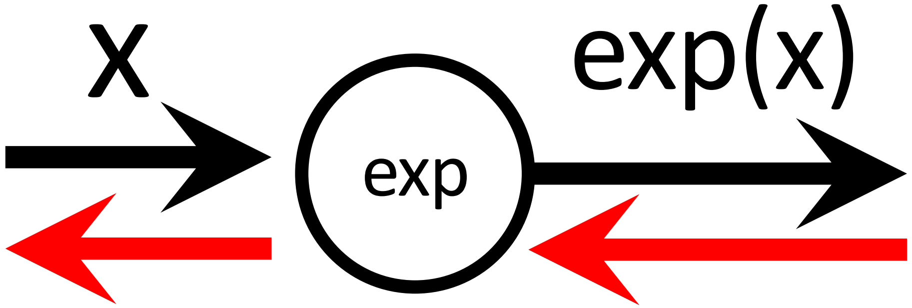
- Sigmoid function : $y = \dfrac{1}{1+\exp(-x)}$

- Computation graph of Sigmoid function

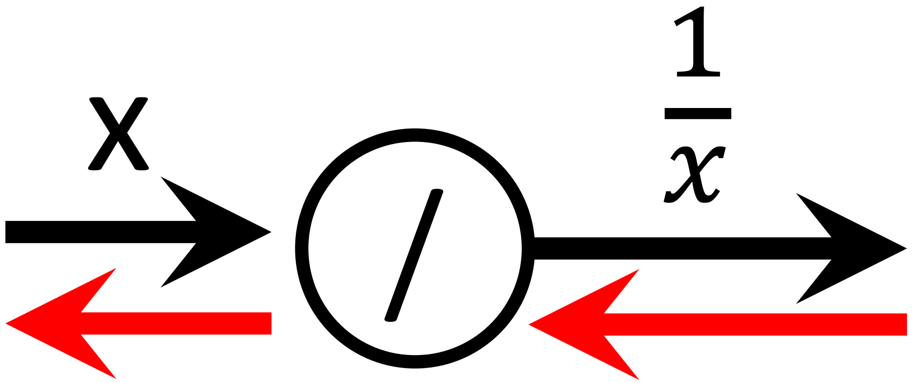


New nodes

- “exp” node



- “/” node



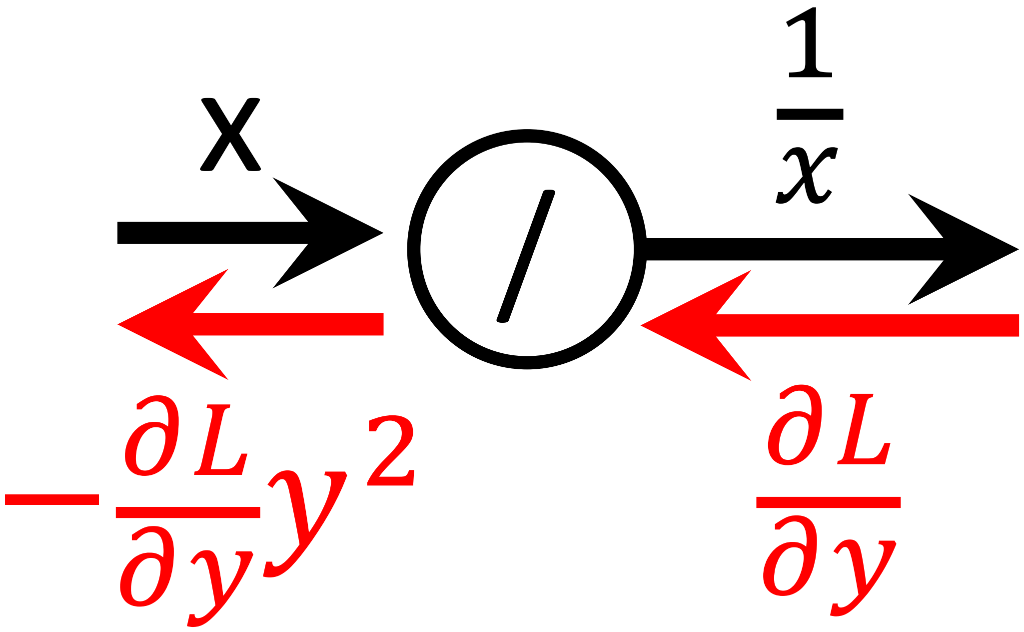
Backward propagation of new node

- "/" node

$$\frac{\partial y}{\partial x} = \frac{\partial}{\partial x} x^{-1}$$

$$= (-1)x^{-2}$$

$$= -\frac{1}{x^2} = -y^2$$

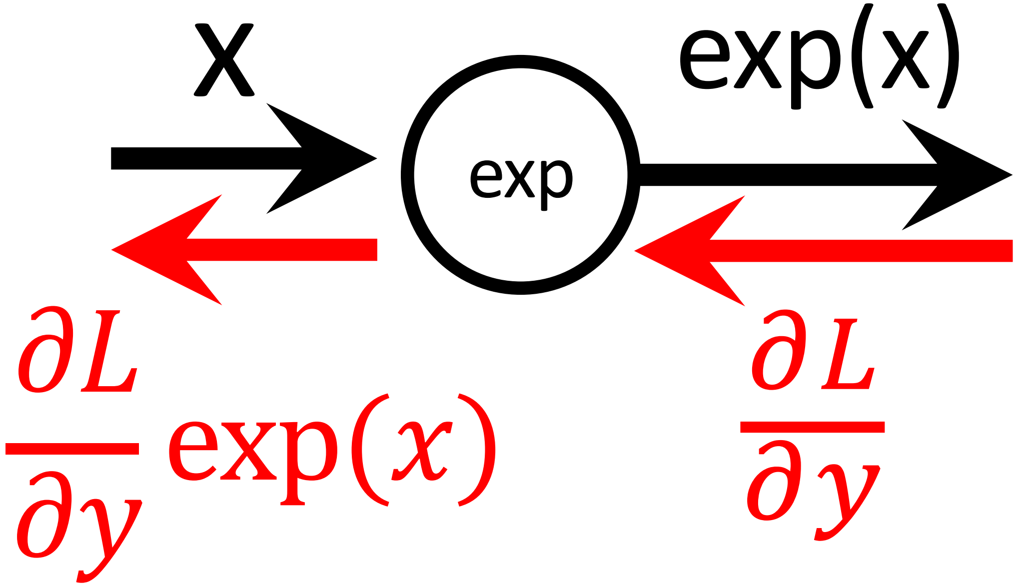


Backward propagation of new node

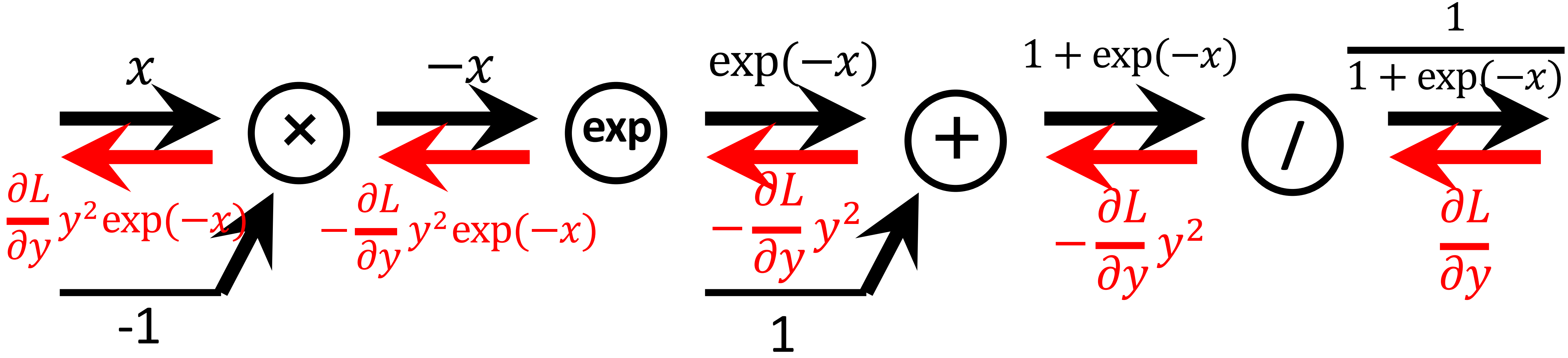
- "exp" node

$$\frac{\partial y}{\partial x} = \frac{\partial}{\partial x}\exp(x)$$

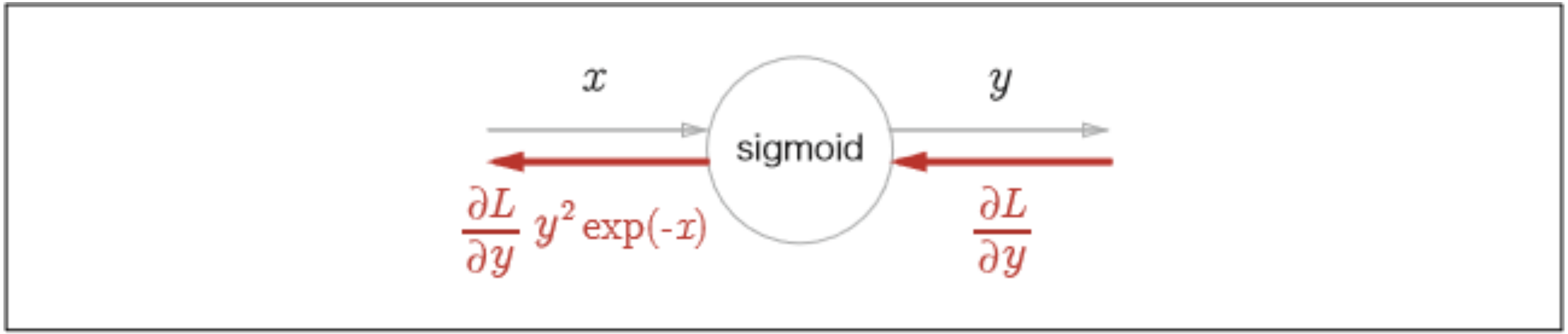
$$= \exp(x)$$



Implementation of Sigmoid layer



Implementation of Sigmoid layer



Implementation of Sigmoid layer

- A differential of Sigmoid function

$$\frac{\partial L}{\partial y} y^2 \exp(-x) = \frac{\partial L}{\partial y} \frac{1}{(1+\exp(-x))^2} \exp(-x)$$

$$= \frac{\partial L}{\partial y} \frac{1}{1+\exp(-x)} \frac{\exp(-x)}{1+\exp(-x)} = \boldsymbol{\frac{\partial L}{\partial y} y(1-y)}$$

- Expressible only by output

Implementation of Sigmoid layer

```
class Sigmoid:
    def __init__(self):
        self.out = None

    def forward(self, x):
        out = 1 / (1 + np.exp(-x))
        self.out = out

        return out

    def backward(self, dout):
        dx = dout * (1.0 - self.out) * self.out

        return dx
```