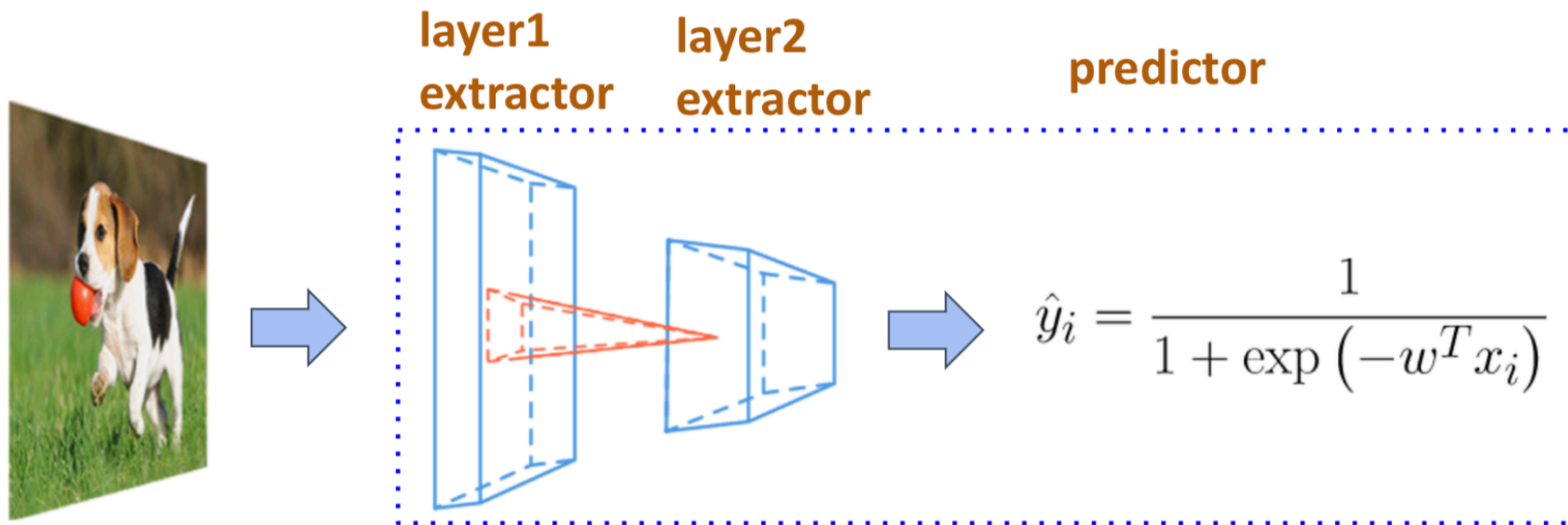


Backpropagation and Automatic Differentiation

Pooyan Jamshidi
UofSC

Model Training Overview



Objective

$$L(w) = \sum_{i=1}^n l(y_i, \hat{y}_i) + \lambda \|w\|^2$$

Training

$$w \leftarrow w - \eta \nabla_w L(w)$$

Symbolic Differentiation

- Input formulae is a symbolic expression tree (computation graph).
- Implement differentiation rules, e.g., sum rule, product rule, chain rule

$$\frac{d(f + g)}{dx} = \frac{df}{dx} + \frac{dg}{dx} \quad \frac{d(fg)}{dx} = \frac{df}{dx}g + f\frac{dg}{dx} \quad \frac{d(h(x))}{dx} = \frac{df(g(x))}{dx} \cdot \frac{dg(x)}{dx}$$

- ✗ For complicated functions, the resultant expression can be exponentially large.
- ✗ Wasteful to keep around intermediate symbolic expressions if we only need a numeric value of the gradient in the end
- ✗ Prone to error

Numerical Differentiation

- We can approximate the gradient using

$$\frac{\partial f(\mathbf{x})}{\partial x_i} \approx \lim_{h \rightarrow 0} \frac{f(\mathbf{x} + h\mathbf{e}_i) - f(\mathbf{x})}{h}$$

$$f(W, x) = W \cdot x$$
$$[-0.8 \quad 0.3] \cdot \begin{bmatrix} 0.5 \\ -0.2 \end{bmatrix}$$

Numerical Differentiation

- We can approximate the gradient using

$$\frac{\partial f(\mathbf{x})}{\partial x_i} \approx \lim_{h \rightarrow 0} \frac{f(\mathbf{x} + h\mathbf{e}_i) - f(\mathbf{x})}{h}$$

$$f(W, x) = W \cdot x \\ [-0.8 \quad 0.3] \cdot \begin{bmatrix} 0.5 \\ -0.2 \end{bmatrix}$$

$$f(W, x) = W \cdot x \\ [-0.8 + \varepsilon \quad 0.3] \cdot \begin{bmatrix} 0.5 \\ -0.2 \end{bmatrix}$$

Numerical Differentiation

- We can approximate the gradient using

$$\frac{\partial f(\mathbf{x})}{\partial x_i} \approx \lim_{h \rightarrow 0} \frac{f(\mathbf{x} + h\mathbf{e}_i) - f(\mathbf{x})}{h}$$

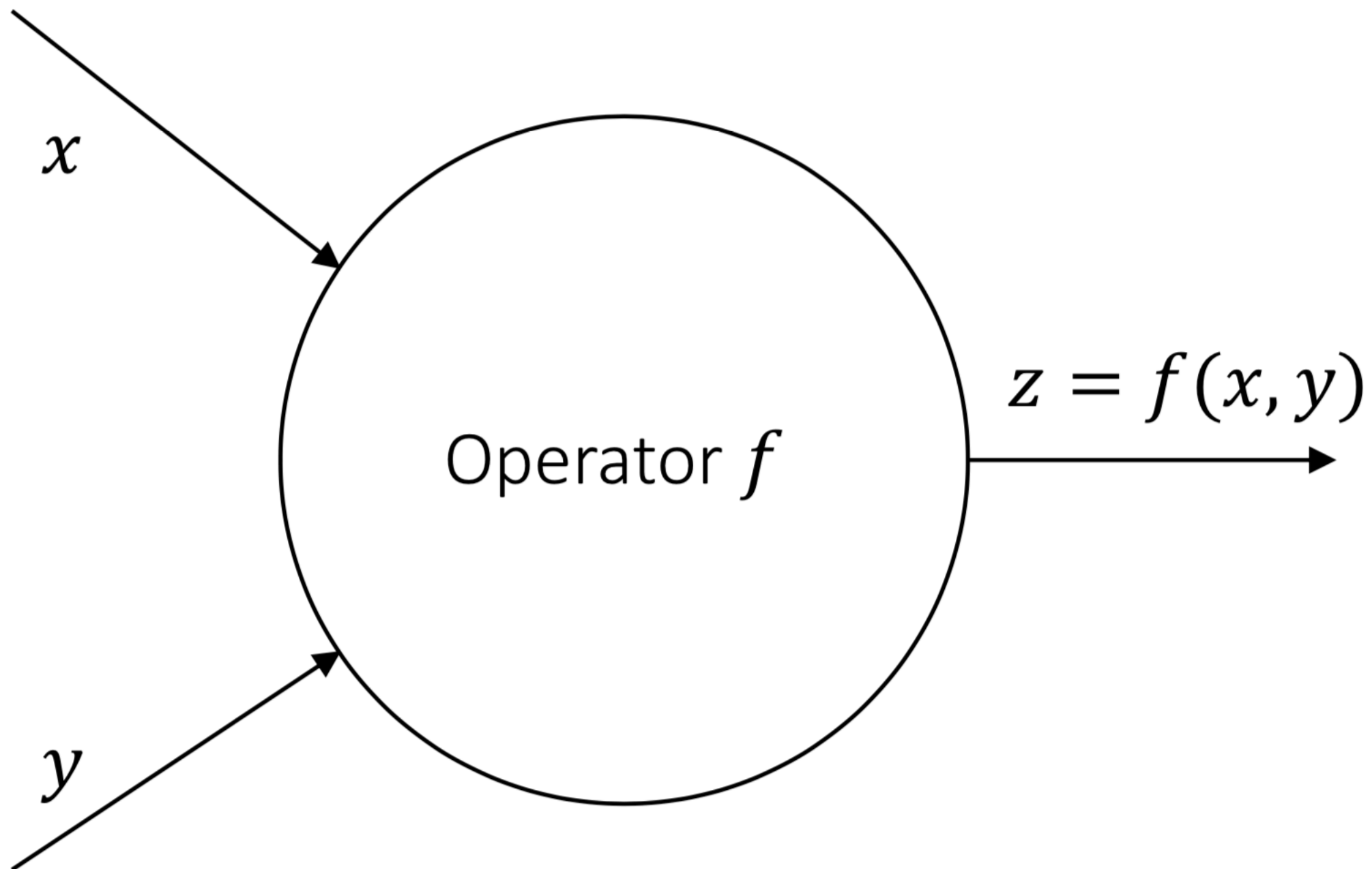
- Reduce the truncation error by using center difference

$$\frac{\partial f(\mathbf{x})}{\partial x_i} \approx \lim_{h \rightarrow 0} \frac{f(\mathbf{x} + h\mathbf{e}_i) - f(\mathbf{x} - h\mathbf{e}_i)}{2h}$$

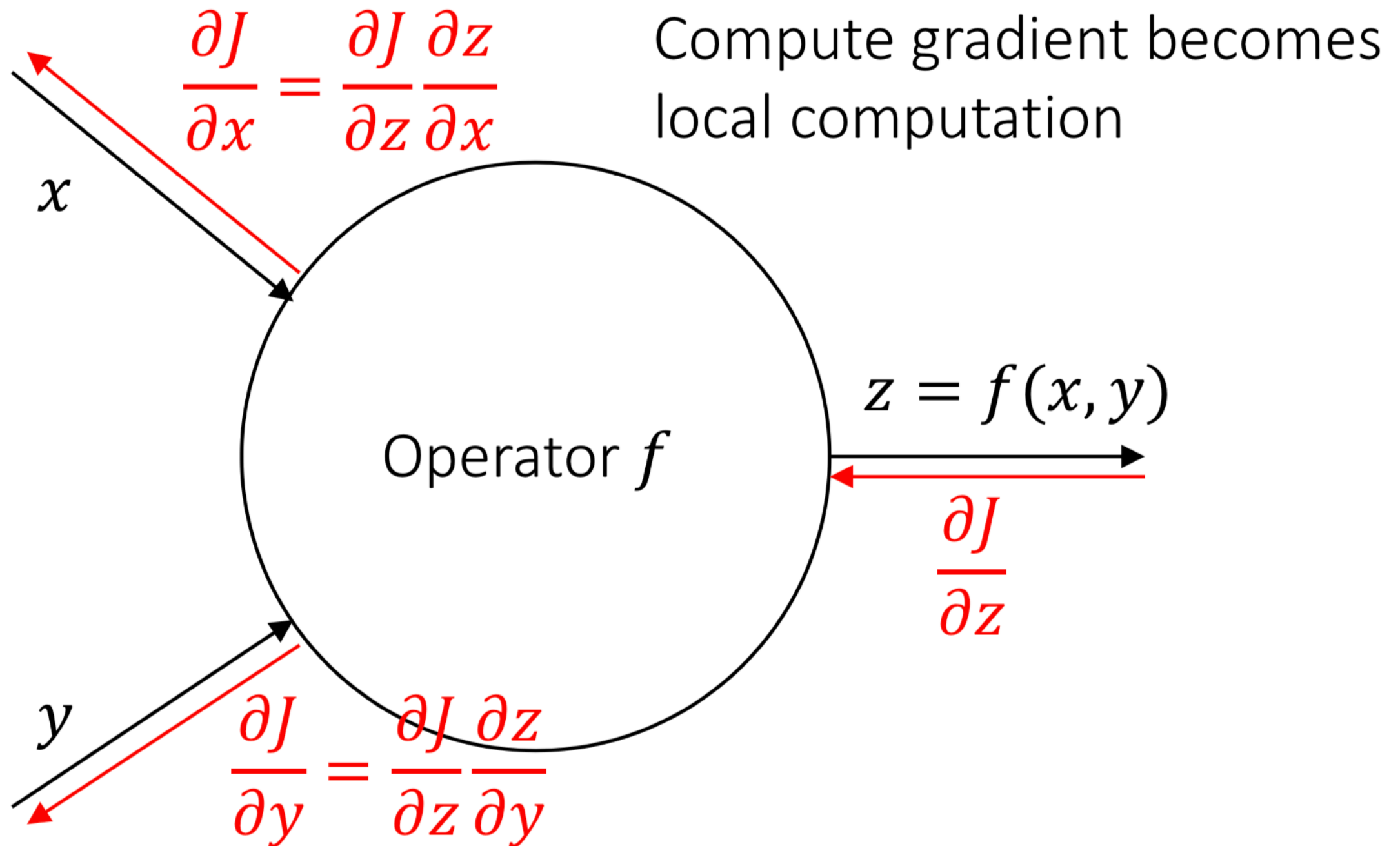
✗ Bad: rounding error, and slow to compute

✓ A powerful tool to check the correctness of implementation, usually use $h = 1\text{e-}6$.

Backpropagation



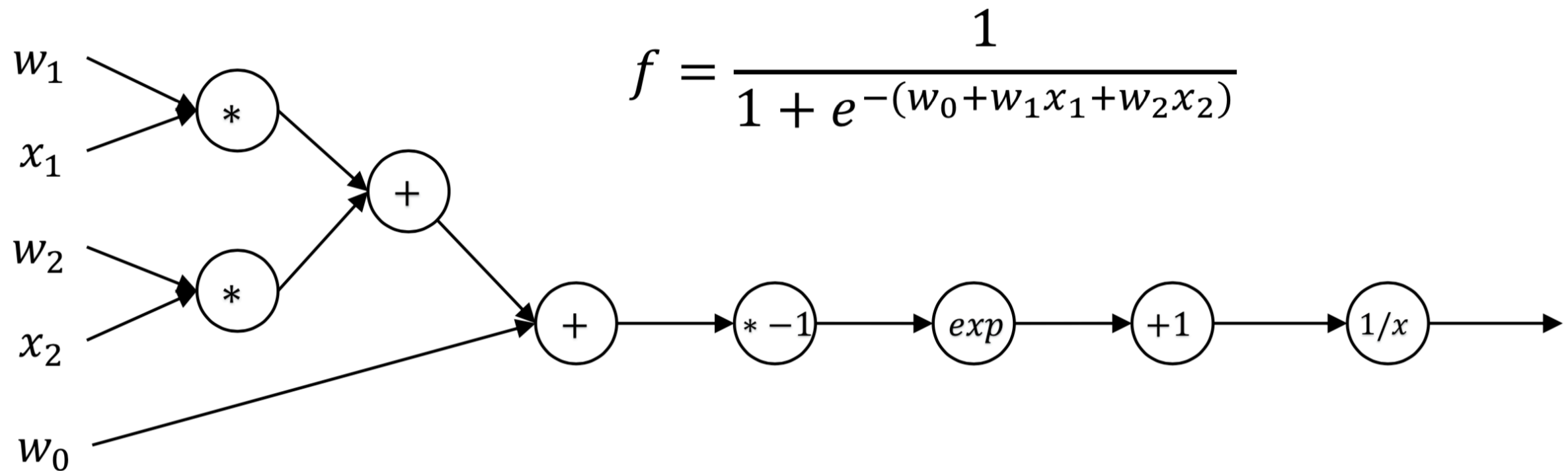
Backpropagation



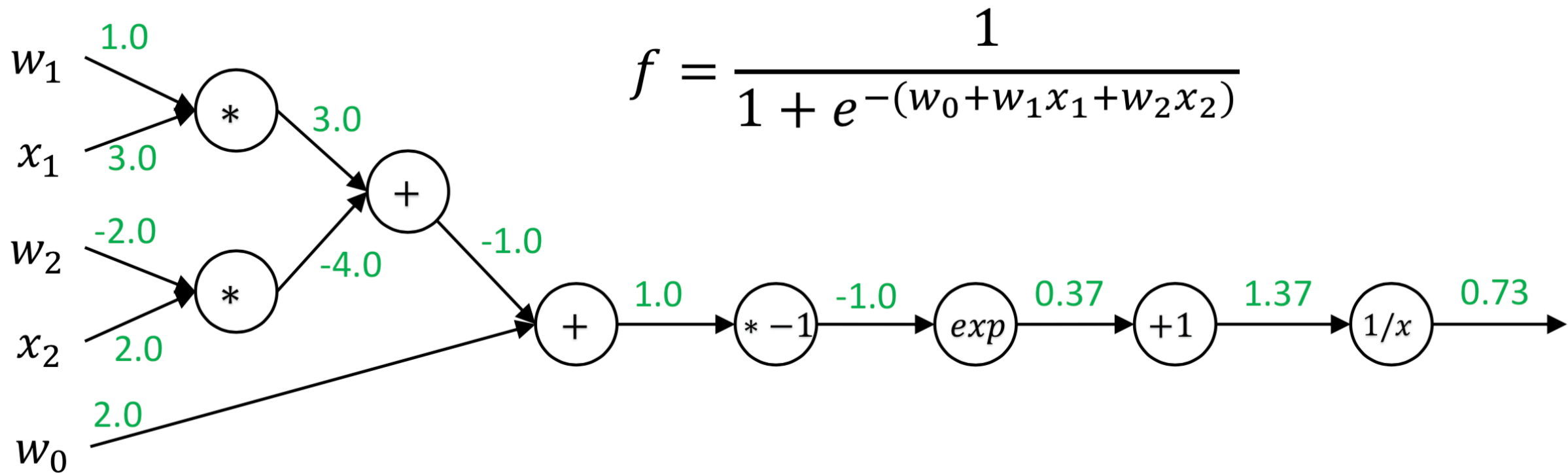
Backpropagation simple example

$$f = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + w_2 x_2)}}$$

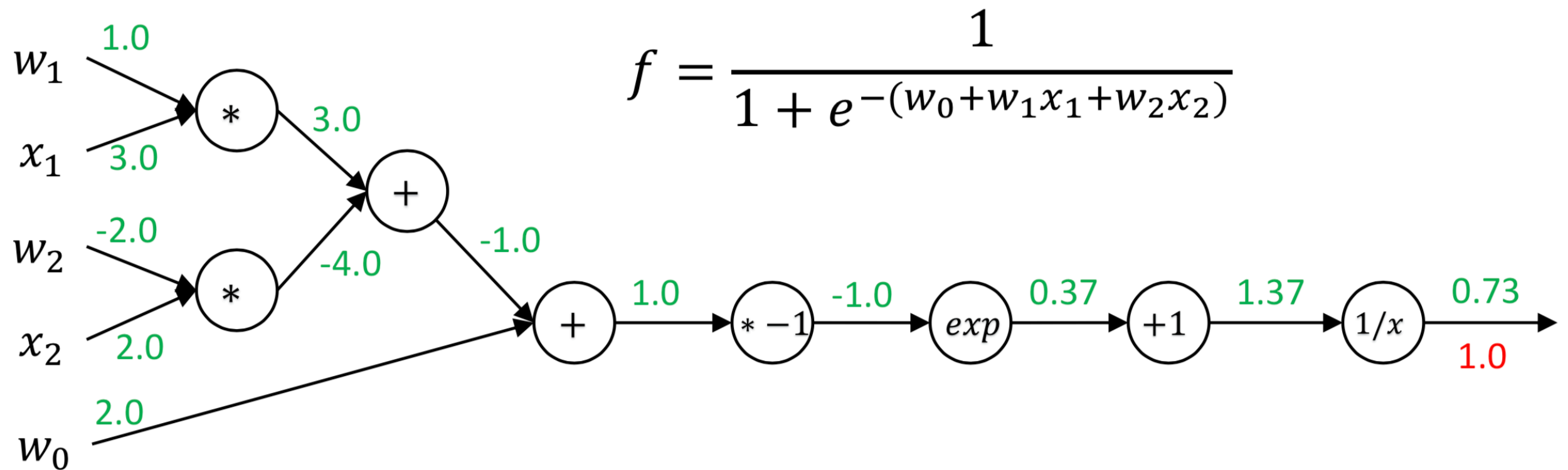
Backpropagation simple example



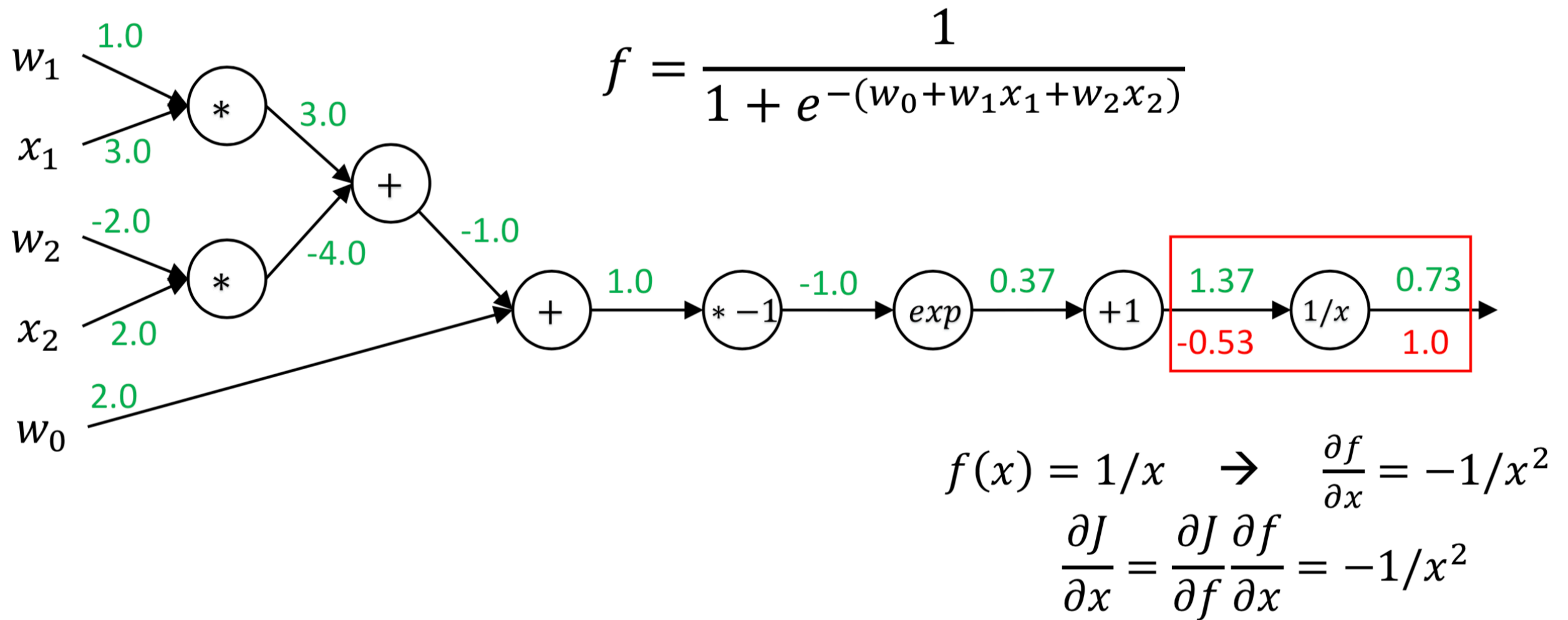
Backpropagation simple example



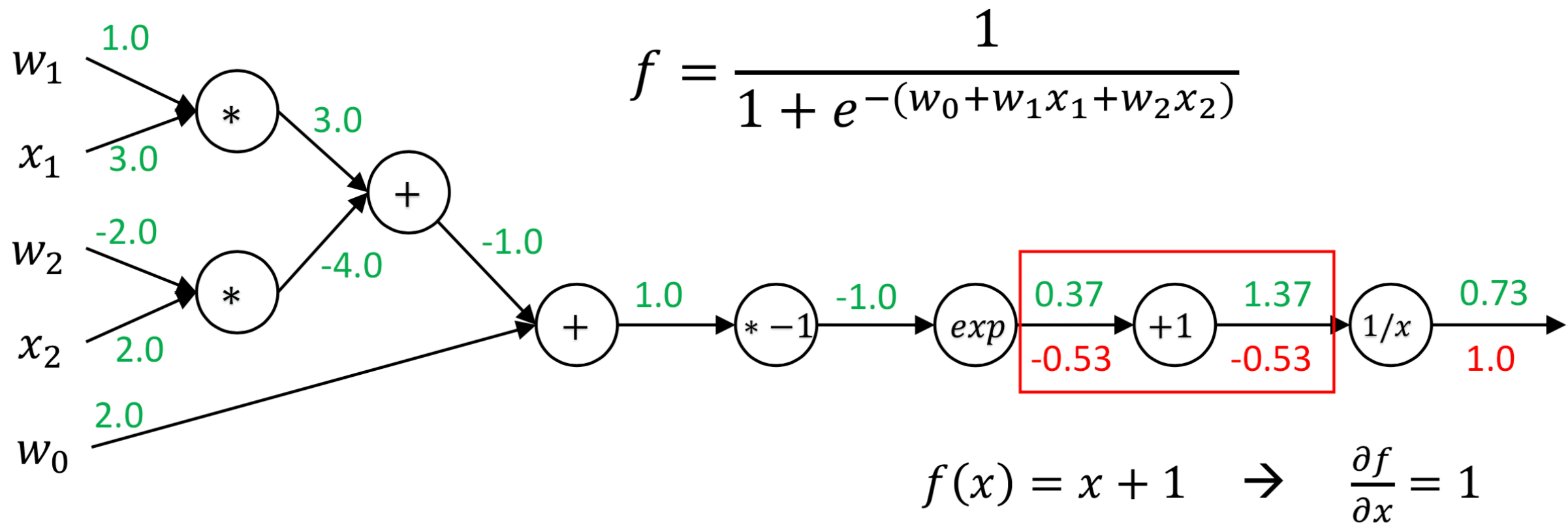
Backpropagation simple example



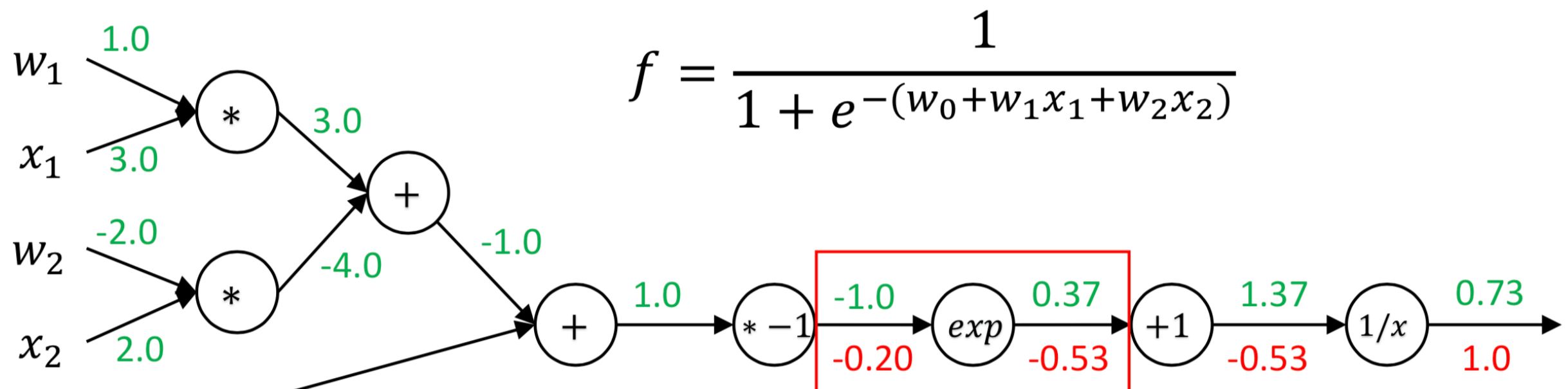
Backpropagation simple example



Backpropagation simple example



Backpropagation simple example

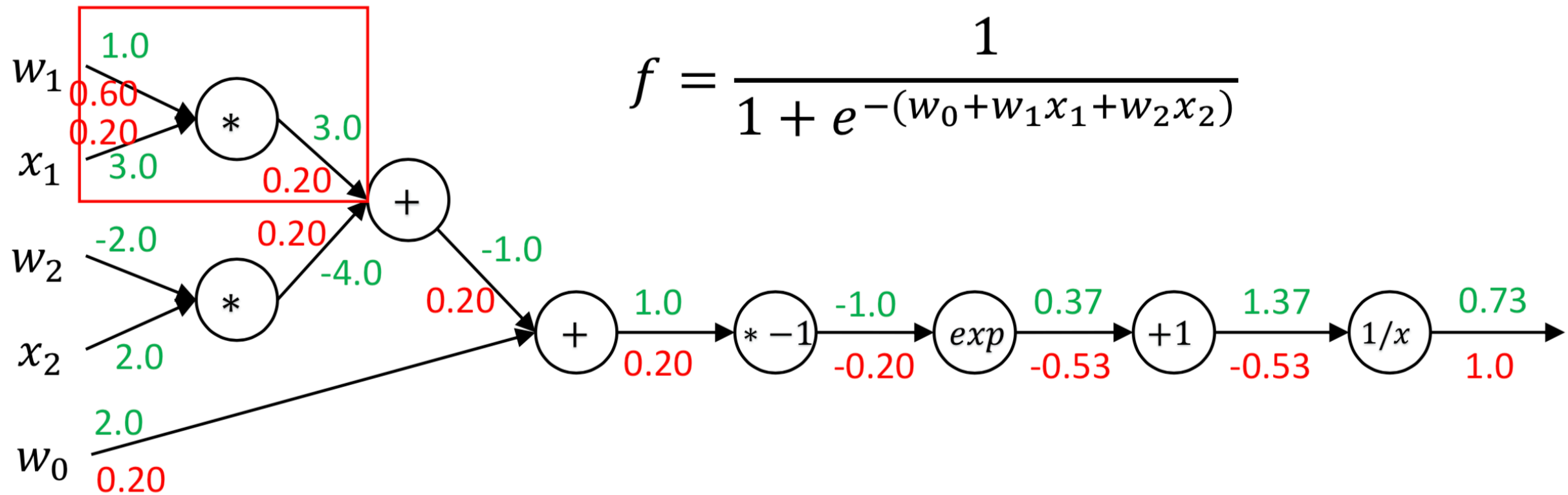


$$f = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + w_2 x_2)}}$$

$$f(x) = e^x \rightarrow \frac{\partial f}{\partial x} = e^x$$

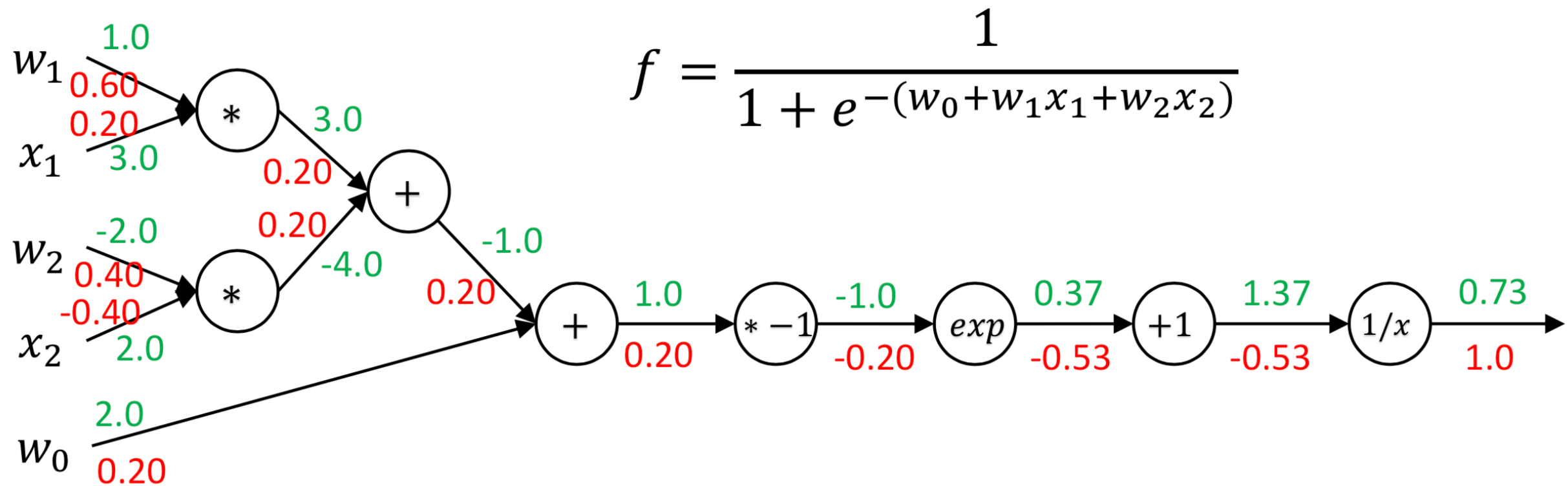
$$\frac{\partial J}{\partial x} = \frac{\partial J}{\partial f} \frac{\partial f}{\partial x} = \frac{\partial J}{\partial f} \cdot e^x$$

Backpropagation simple example



$$f(x, w) = xw \quad \rightarrow \quad \frac{\partial f}{\partial x} = w, \quad \frac{\partial f}{\partial w} = x$$

Backpropagation simple example



**Any problem? Can we
do better?**

Problems of backpropagation

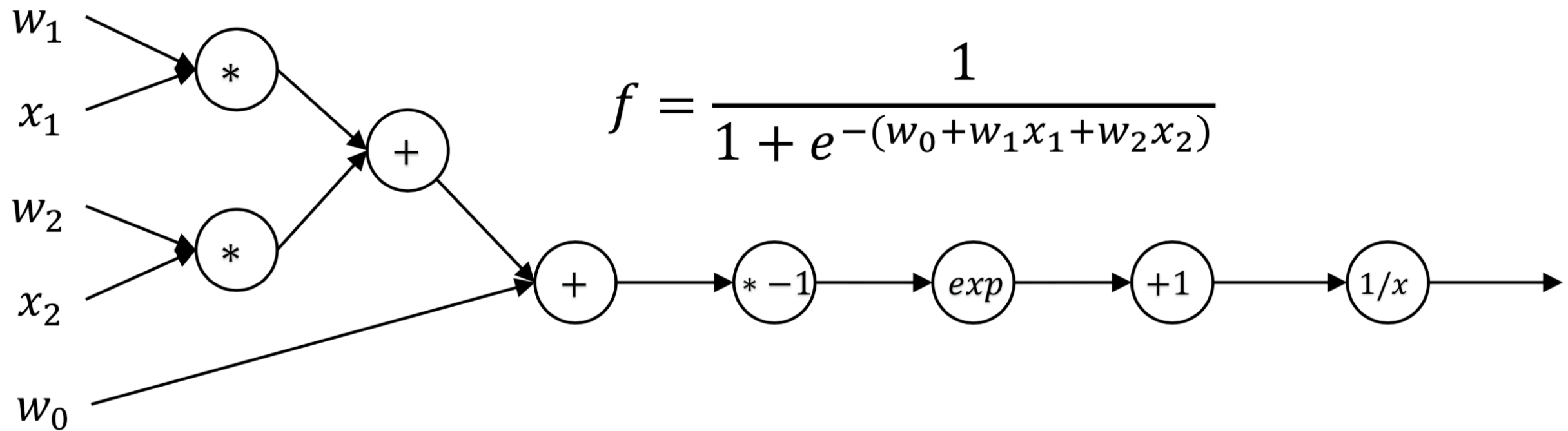
- You always need to keep intermediate data in the memory during the forward pass in case it will be used in the backpropagation.
- Lack of flexibility, e.g., compute the gradient of gradient.

Automatic Differentiation (autodiff)

- Create computation graph for gradient computation

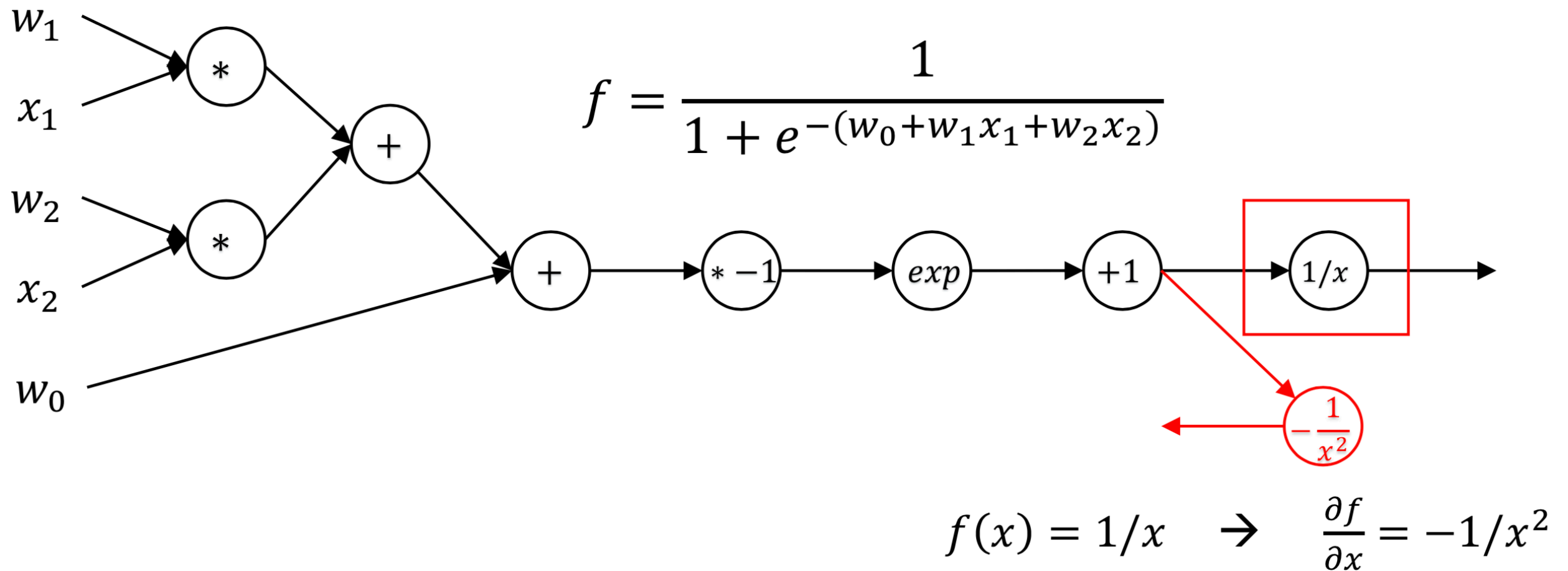
Automatic Differentiation (autodiff)

- Create computation graph for gradient computation



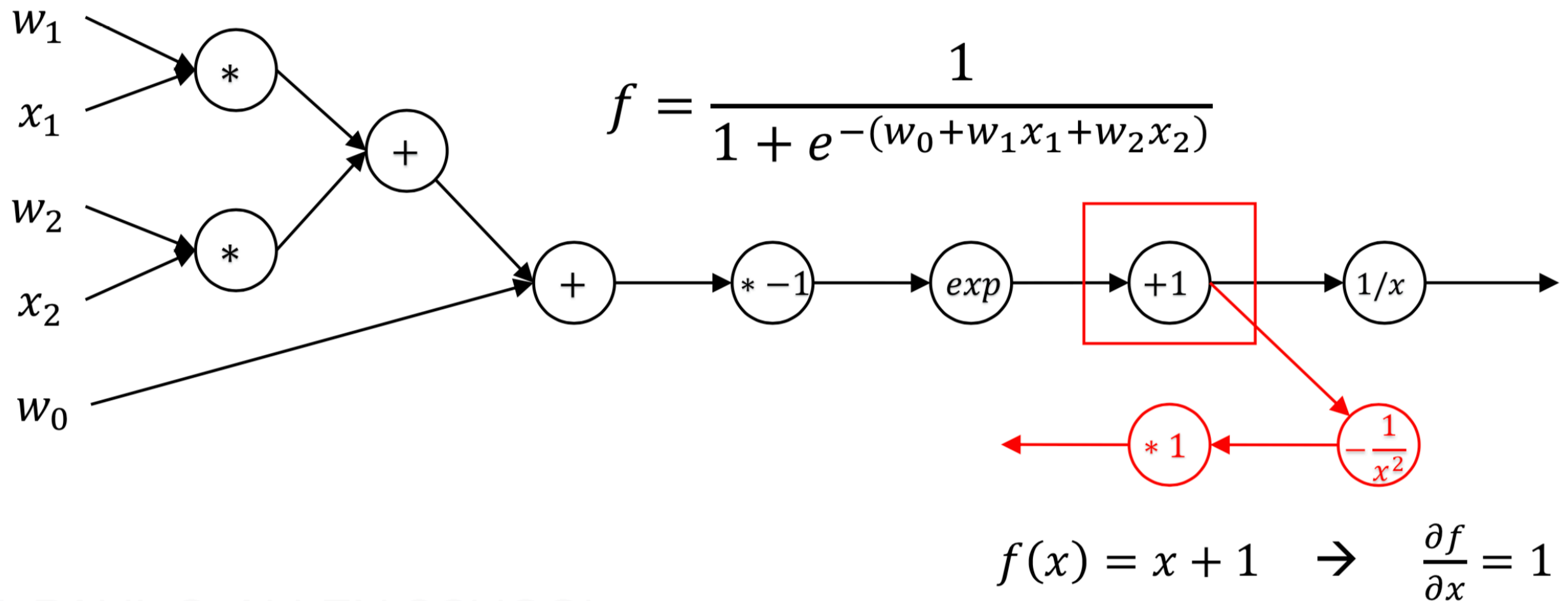
Automatic Differentiation (autodiff)

- Create computation graph for gradient computation



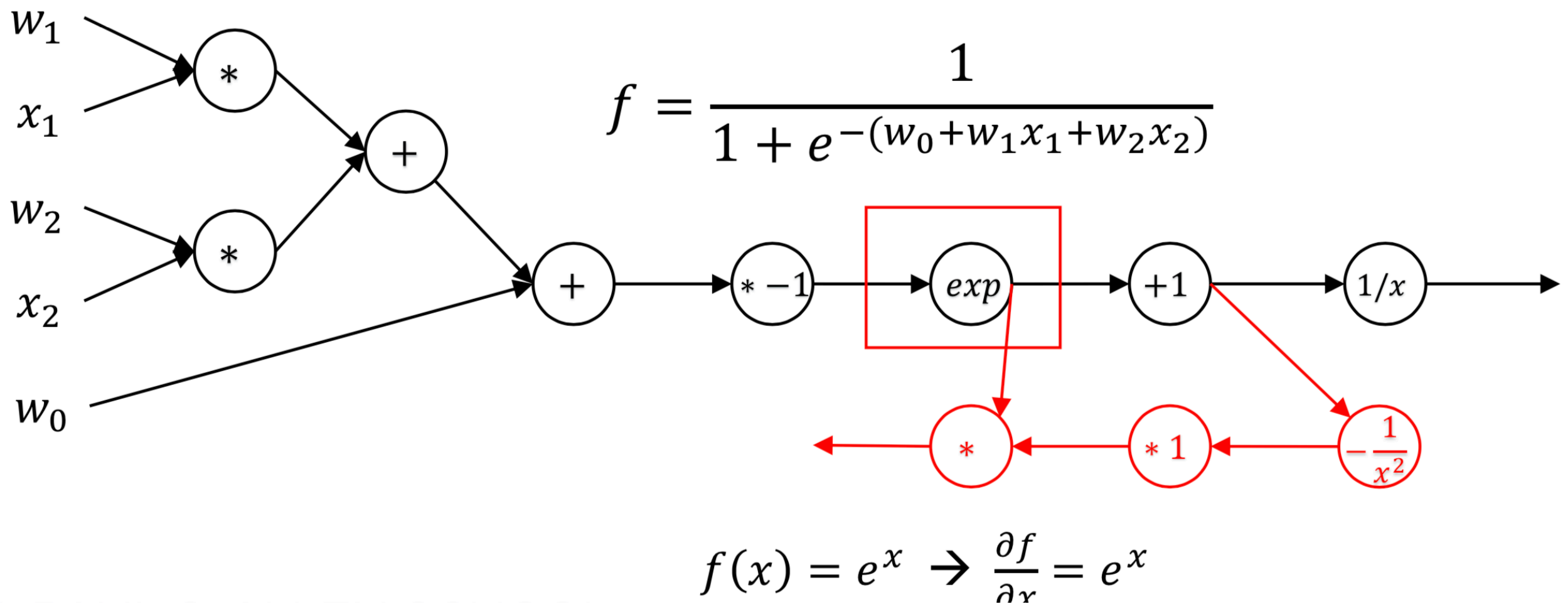
Automatic Differentiation (autodiff)

- Create computation graph for gradient computation



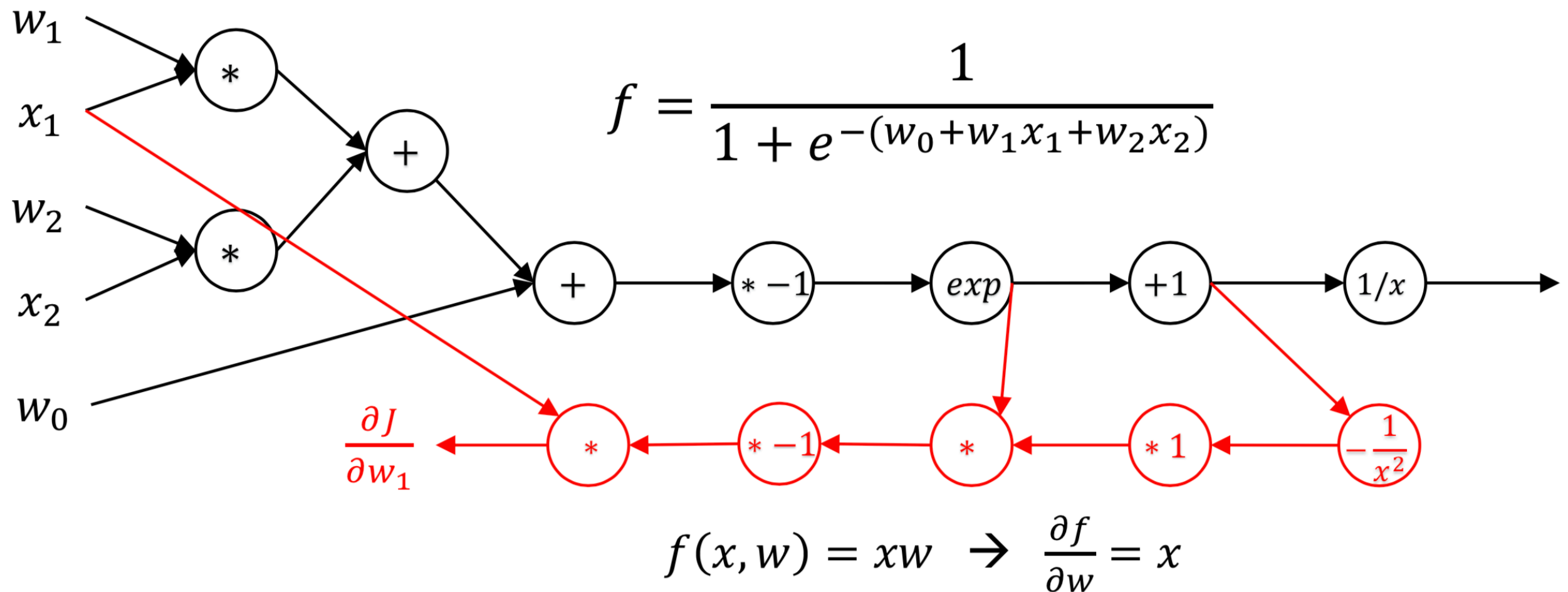
Automatic Differentiation (autodiff)

- Create computation graph for gradient computation



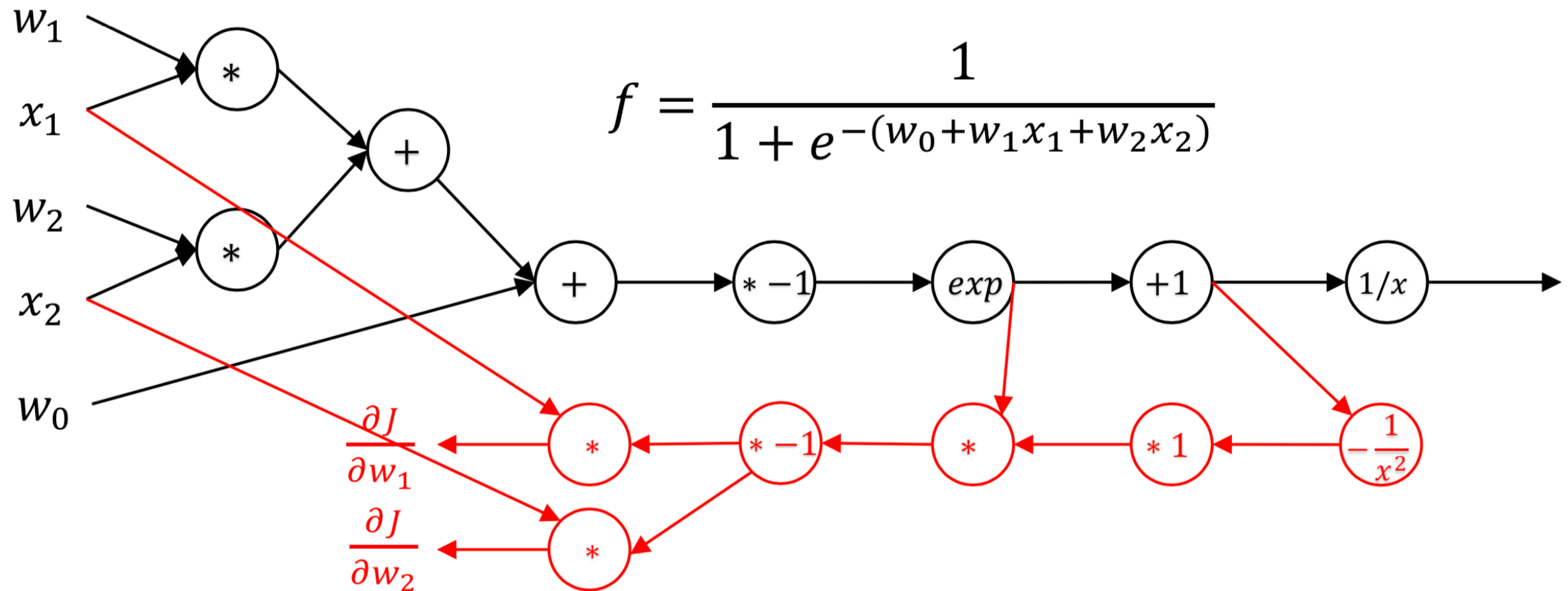
Automatic Differentiation (autodiff)

- Create computation graph for gradient computation

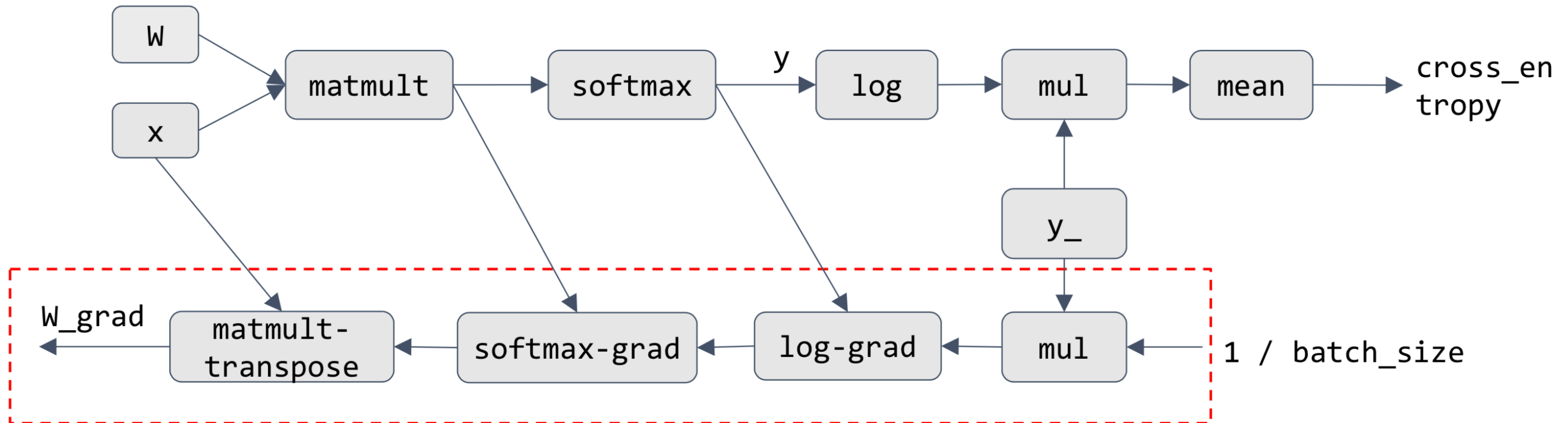


Automatic Differentiation (autodiff)

- Create computation graph for gradient computation

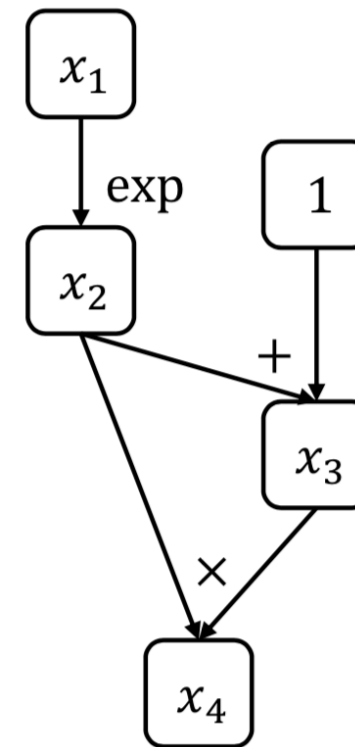


AutoDiff Algorithm



AutoDiff Algorithm

⇒ def gradient(out):
 node_to_grad[out] = 1
 nodes = get_node_list(out)
 for node in reverse_topo_order(nodes):
 grad ← sum partial adjoints from output edges
 input_grads ← node.op.gradient(input, grad) for
 input in node.inputs
 add input_grads to node_to_grad
 return node_to_grad



AutoDiff Algorithm

```
def gradient(out):
```



```
    node_to_grad[out] = 1
```

```
    nodes = get_node_list(out)
```

```
    for node in reverse_topo_order(nodes):
```

```
        grad ← sum partial adjoints from output edges
```

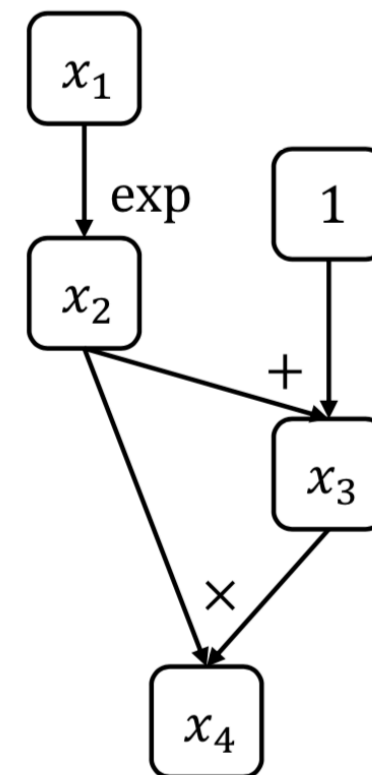
```
        input_grads ← node.op.gradient(input, grad) for  
input in node.inputs
```

```
        add input_grads to node_to_grad
```

```
    return node_to_grad
```

```
node_to_grad:
```

```
     $x_4$ :  $\overline{x_4}$ 
```



$\overline{x_4}$

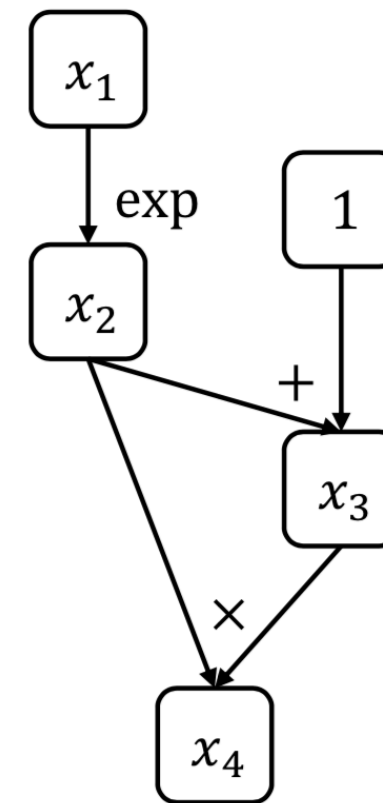
AutoDiff Algorithm

```
def gradient(out):  
    node_to_grad[out] = 1  
    nodes = get_node_list(out)  
    ⇒ for node in reverse_topo_order(nodes):  
        grad ← sum partial adjoints from output edges  
        input_grads ← node.op.gradient(input, grad) for  
        input in node.inputs  
        add input_grads to node_to_grad  
    return node_to_grad
```

node_to_grad:

$x_4: \overline{x_4}$

x_4	x_3	x_2	x_1
-------	-------	-------	-------



$\overline{x_4}$

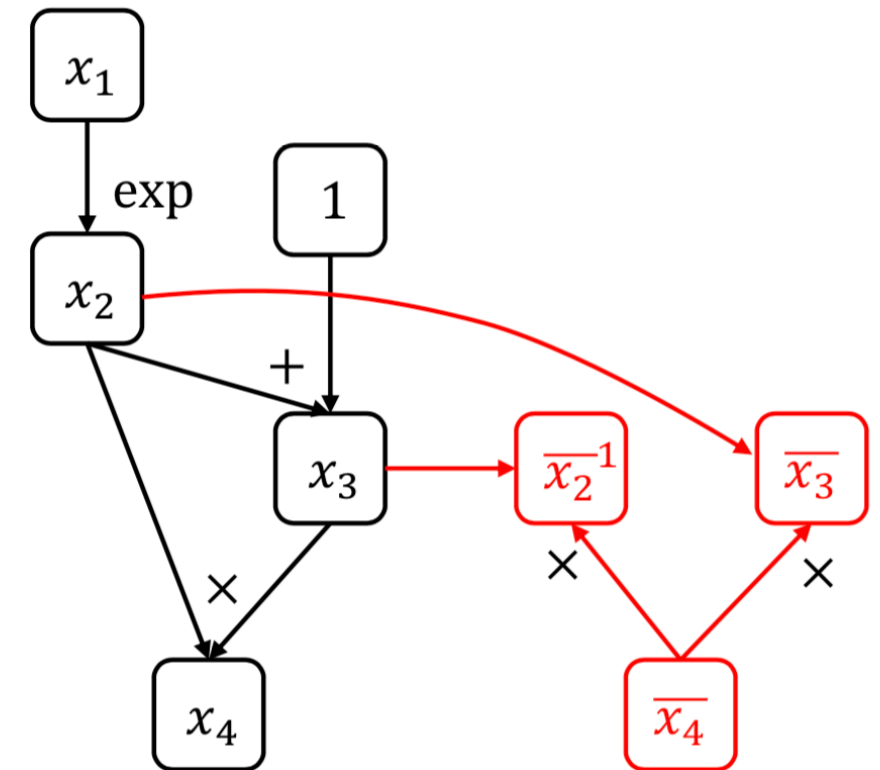
AutoDiff Algorithm

```
def gradient(out):  
    node_to_grad[out] = 1  
    nodes = get_node_list(out)  
    for node in reverse_topo_order(nodes):  
        grad ← sum partial adjoints from output edges  
        input_grads ← node.op.gradient(input, grad) for  
        input in node.inputs  
        add input_grads to node_to_grad  
    return node_to_grad
```

node_to_grad:

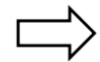
$x_4: \overline{x_4}$

x_4	x_3	x_2	x_1
-------	-------	-------	-------



AutoDiff Algorithm

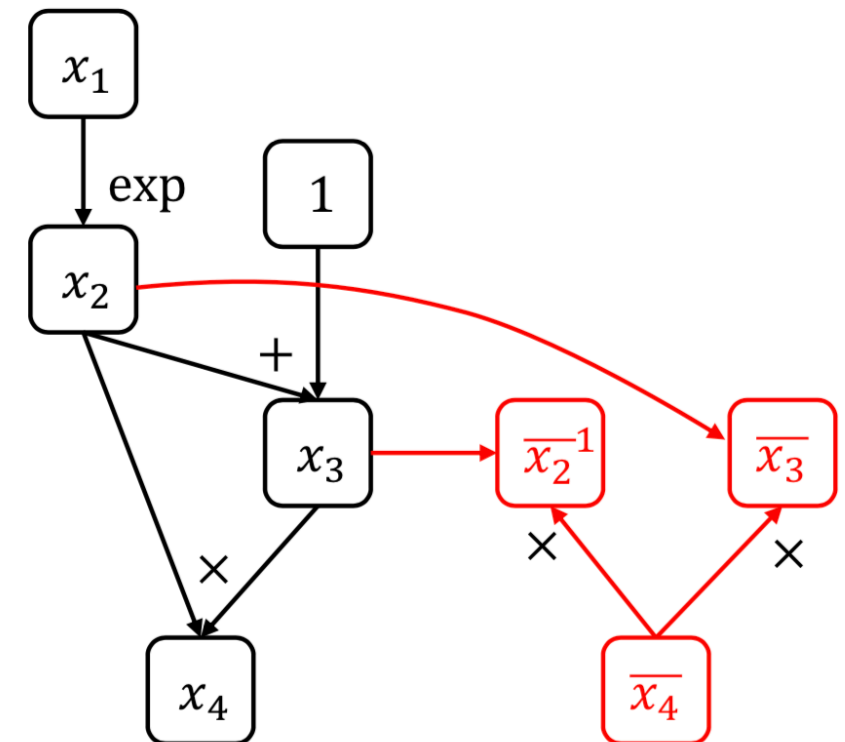
```
def gradient(out):  
    node_to_grad[out] = 1  
    nodes = get_node_list(out)  
    for node in reverse_topo_order(nodes):  
        grad ← sum partial adjoints from output edges  
        input_grads ← node.op.gradient(input, grad) for  
        input in node.inputs  
    add input_grads to node_to_grad  
    return node_to_grad
```



node_to_grad:

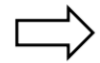
$x_4: \overline{x_4}$
 $x_3: \overline{x_3}$
 $x_2: \overline{x_2}^{-1}$

x_4	x_3	x_2	x_1
-------	-------	-------	-------



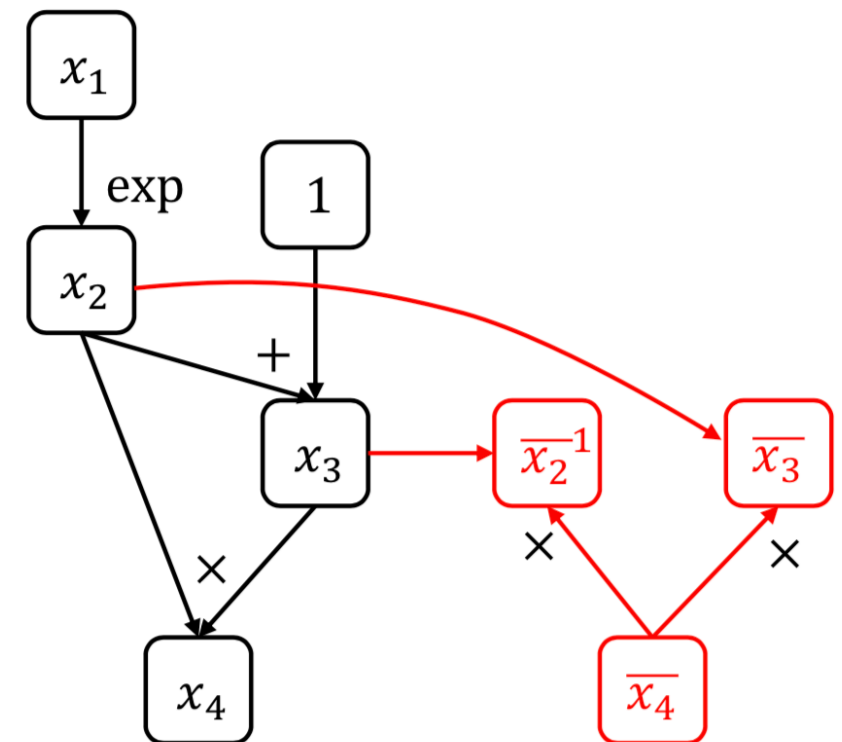
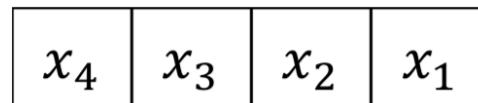
AutoDiff Algorithm

```
def gradient(out):  
    node_to_grad[out] = 1  
    nodes = get_node_list(out)  
    for node in reverse_topo_order(nodes):  
        grad ← sum partial adjoints from output edges  
        input_grads ← node.op.gradient(input, grad) for  
input in node.inputs  
        add input_grads to node_to_grad  
    return node_to_grad
```



node_to_grad:

$x_4: \overline{x_4}$
 $x_3: \overline{x_3}$
 $x_2: \overline{x_2}^{-1}$



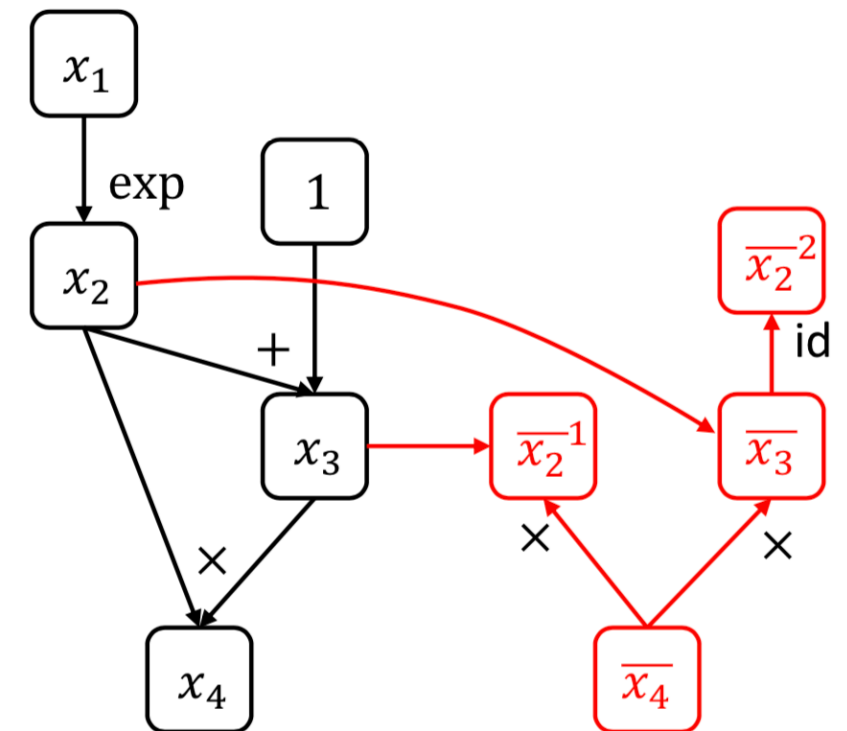
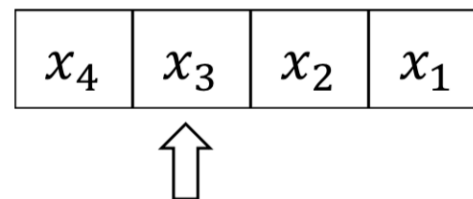
AutoDiff Algorithm

```
def gradient(out):  
    node_to_grad[out] = 1  
    nodes = get_node_list(out)  
    for node in reverse_topo_order(nodes):  
        grad ← sum partial adjoints from output edges  
        input_grads ← node.op.gradient(input, grad) for  
input in node.inputs  
        add input_grads to node_to_grad  
    return node_to_grad
```



node_to_grad:

$x_4: \overline{x_4}$
 $x_3: \overline{x_3}$
 $x_2: \overline{x_2}^1$

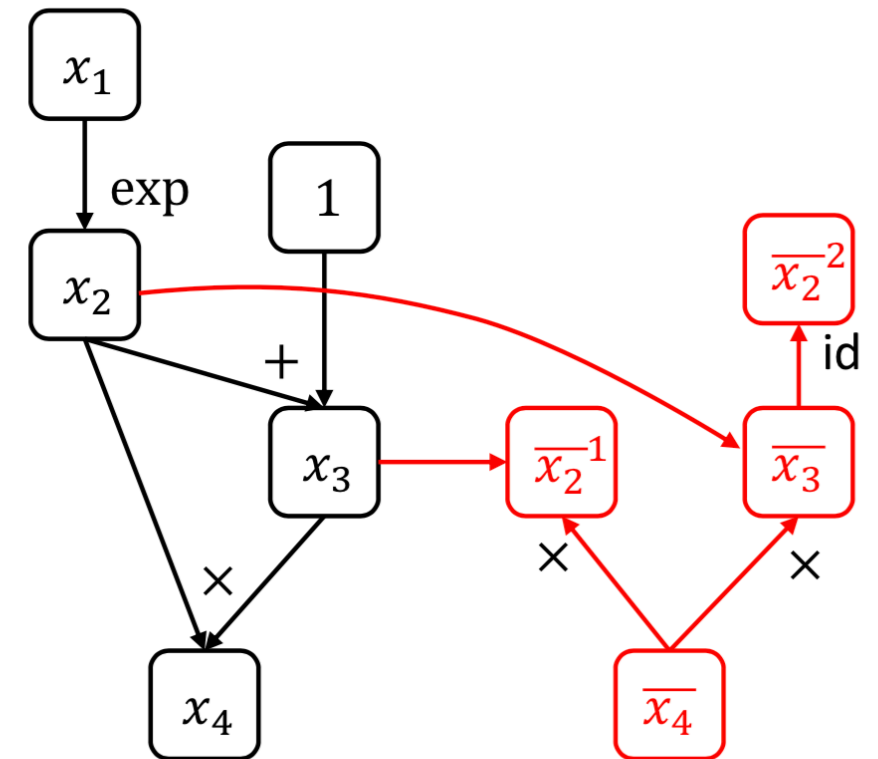
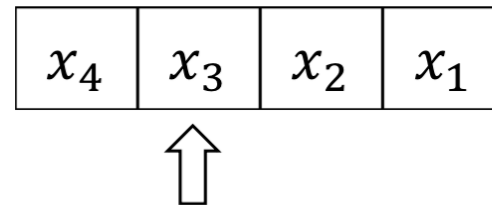


AutoDiff Algorithm

```
def gradient(out):  
    node_to_grad[out] = 1  
    nodes = get_node_list(out)  
    for node in reverse_topo_order(nodes):  
        grad ← sum partial adjoints from output edges  
        input_grads ← node.op.gradient(input, grad) for  
        input in node.inputs  
    ⇒ add input_grads to node_to_grad  
    return node_to_grad
```

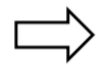
node_to_grad:

x_4 : $\overline{x_4}$
 x_3 : $\overline{x_3}$
 x_2 : $\overline{x_2}^1, \overline{x_2}^2$



AutoDiff Algorithm

```
def gradient(out):
    node_to_grad[out] = 1
    nodes = get_node_list(out)
    for node in reverse_topo_order(nodes):
        grad ← sum partial adjoints from output edges
        input_grads ← node.op.gradient(input, grad) for
input in node.inputs
        add input_grads to node_to_grad
    return node_to_grad
```

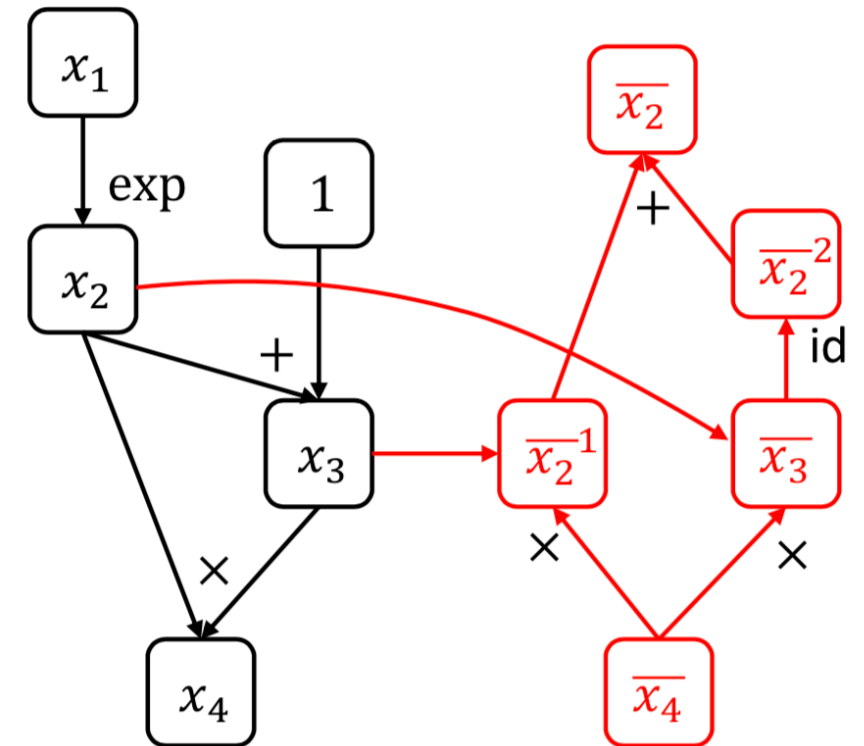
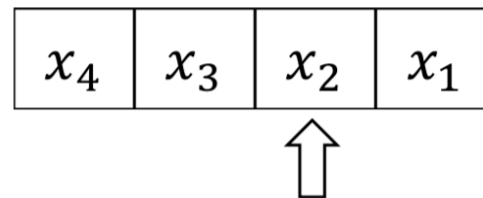


node_to_grad:

x_4 : $\overline{x_4}$

x_3 : $\overline{x_3}$

x_2 : $\overline{x_2}^1, \overline{x_2}^2$

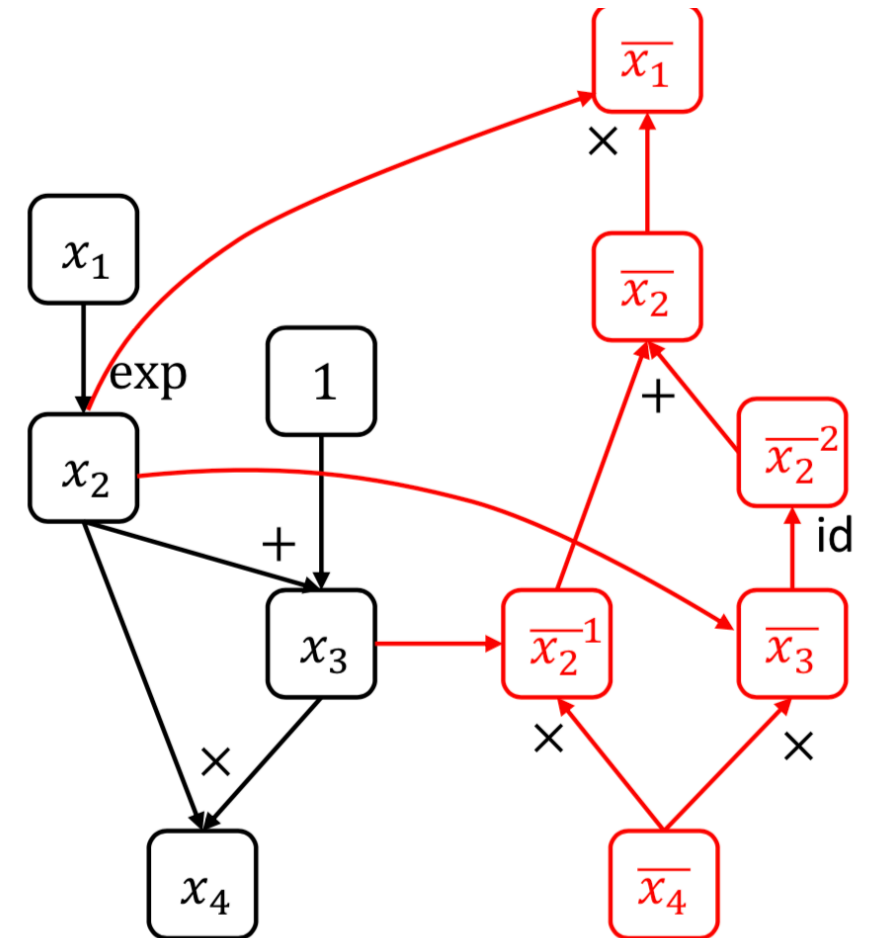
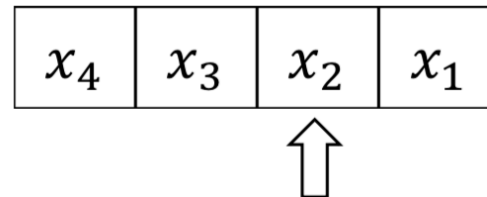


AutoDiff Algorithm

```
def gradient(out):  
    node_to_grad[out] = 1  
    nodes = get_node_list(out)  
    for node in reverse_topo_order(nodes):  
        grad ← sum partial adjoints from output edges  
        input_grads ← node.op.gradient(input, grad) for  
        input in node.inputs  
        add input_grads to node_to_grad  
    return node_to_grad
```

node_to_grad:

x_4 : $\overline{x_4}$
 x_3 : $\overline{x_3}$
 x_2 : $\overline{x_2}^1$, $\overline{x_2}^2$

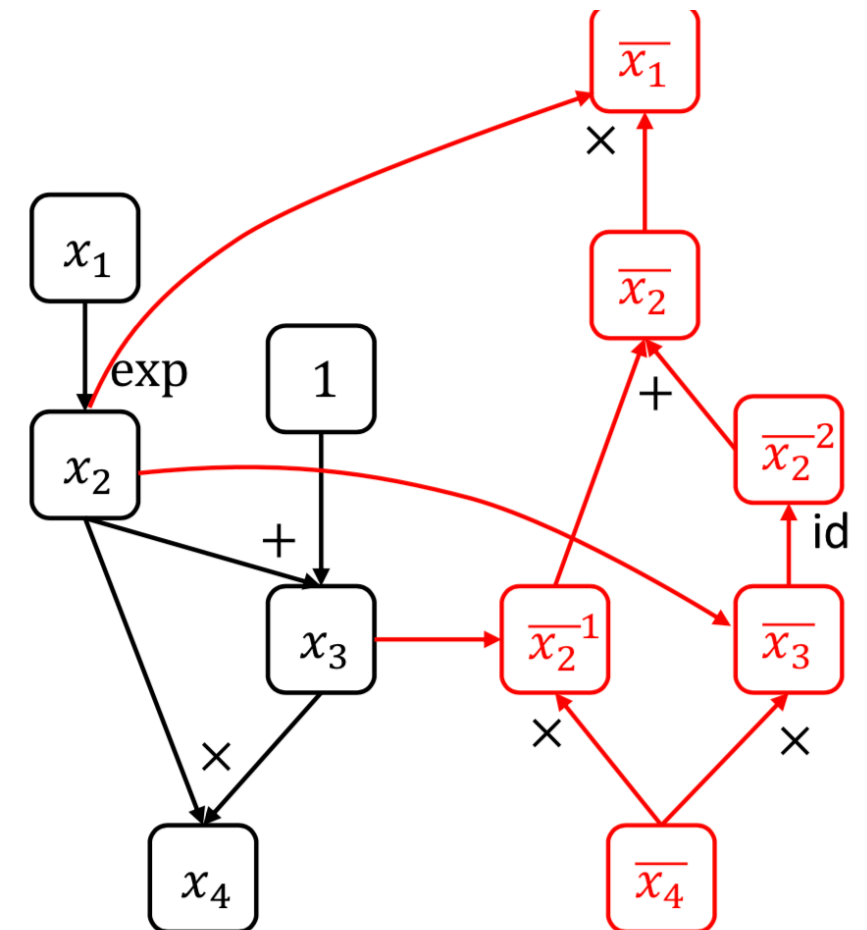
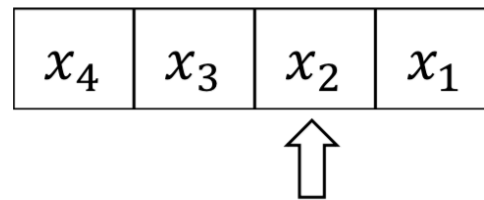


AutoDiff Algorithm

```
def gradient(out):
    node_to_grad[out] = 1
    nodes = get_node_list(out)
    for node in reverse_topo_order(nodes):
        grad ← sum partial adjoints from output edges
        input_grads ← node.op.gradient(input, grad) for
        input in node.inputs
    → add input_grads to node_to_grad
    return node_to_grad
```

node_to_grad:

$x_4: \overline{x_4}$
 $x_3: \overline{x_3}$
 $x_2: \overline{x_2}^1, \overline{x_2}^2$
 $x_1: \overline{x_1}$



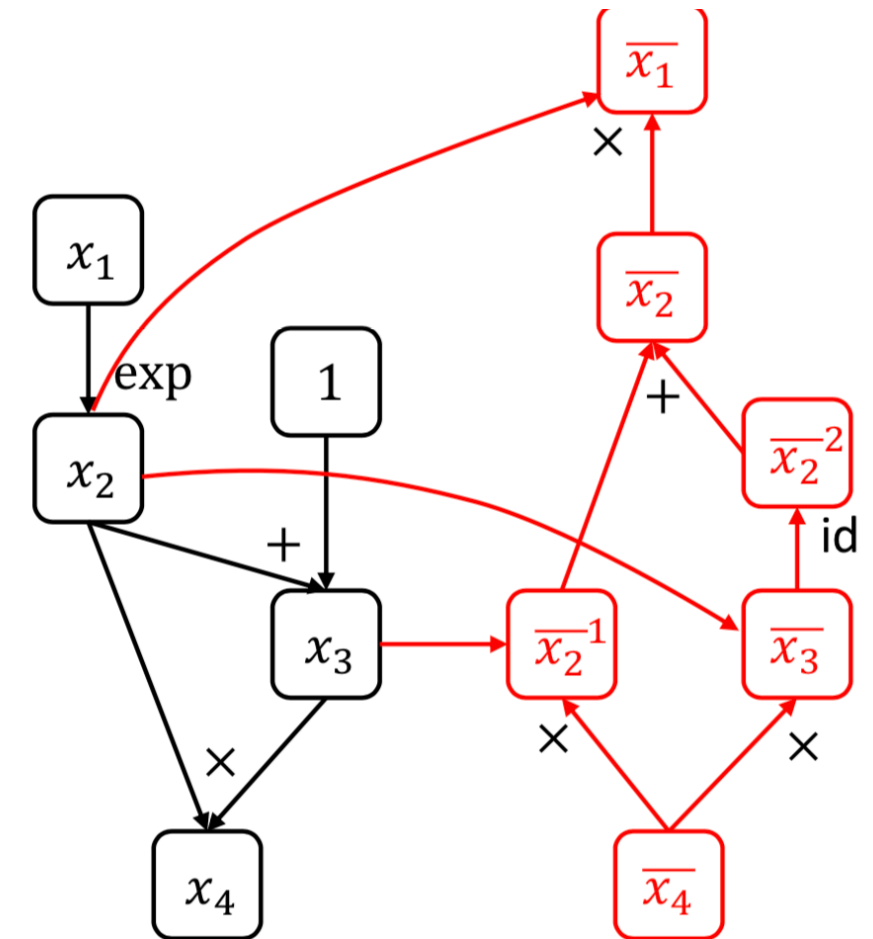
AutoDiff Algorithm

```
def gradient(out):
    node_to_grad[out] = 1
    nodes = get_node_list(out)
    for node in reverse_topo_order(nodes):
        grad ← sum partial adjoints from output edges
        input_grads ← node.op.gradient(input, grad) for
input in node.inputs
        add input_grads to node_to_grad
    return node_to_grad
```

node_to_grad:

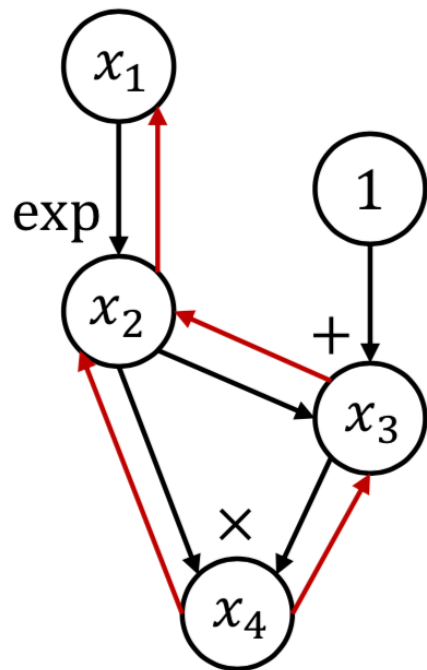
$x_4: \overline{x_4}$
 $x_3: \overline{x_3}$
 $x_2: \overline{x_2}^1, \overline{x_2}^2$
 $x_1: \overline{x_1}$

x_4	x_3	x_2	x_1
-------	-------	-------	-------

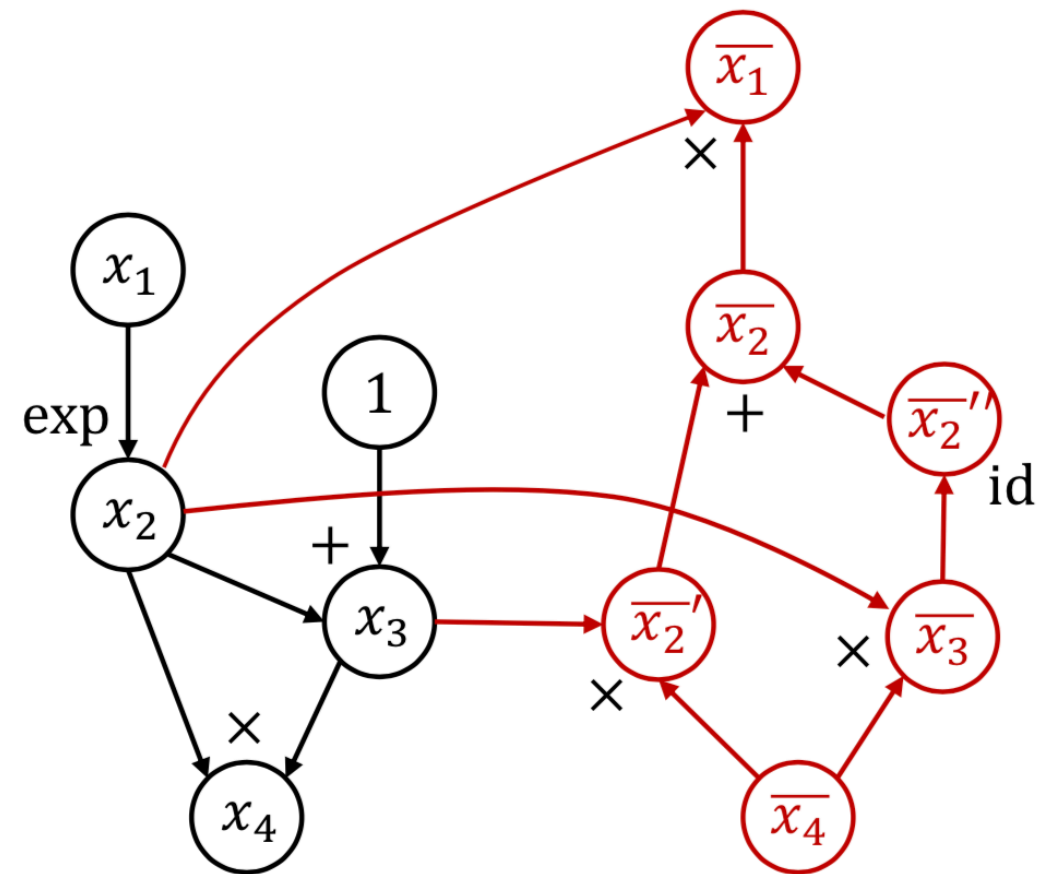


Backpropagation vs AutoDiff

Backpropagation



AutoDiff



Recap

- Numerical differentiation
 - Tool to check the correctness of implementation
- Backpropagation
 - Easy to understand and implement
 - Bad for memory use and schedule optimization
- Automatic differentiation
 - Generate gradient computation to entire computation graph
 - Better for system optimization

References

- Automatic differentiation in machine learning: a survey <https://arxiv.org/abs/1502.05767>
- CS231n backpropagation: <http://cs231n.github.io/optimization-2/>