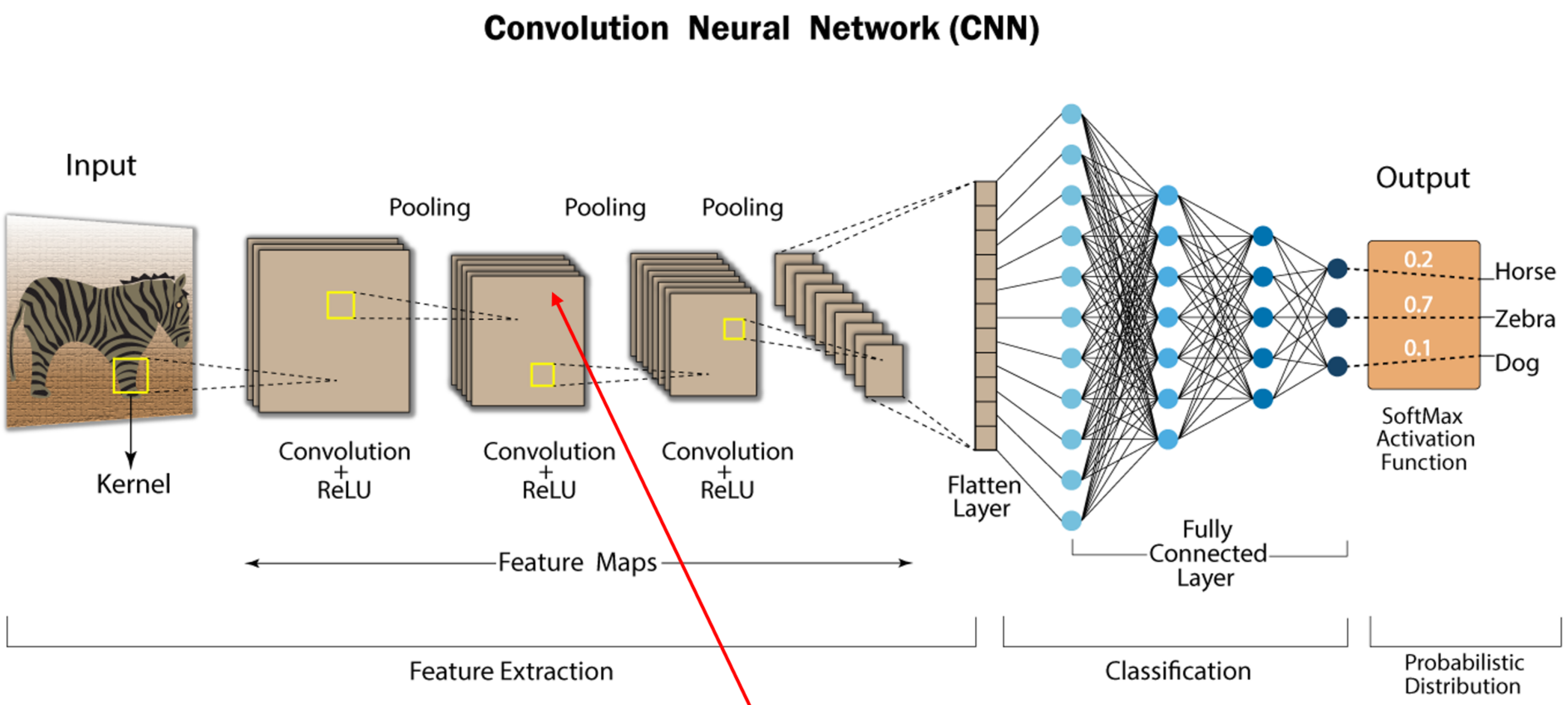
COMPSCI 389
Introduction to Machine Learning

Days: Tu/Th. **Time:** 2:30 – 3:45 **Building:** Morrill 2 **Room:** 222

Topic 10.0: Automatic Differentiation

Prof. Philip S. Thomas (pthomas@cs.umass.edu)

Coming up...



To train the model, we need the derivative of the loss function with respect to each weight.
How can we compute the derivative with respect to this weight in the model?

Old Answer: Manual Calculus!

- By finding clever patters in the derivatives, they can be derived and computed **relatively** easily.
  - … for fully connected feed forward networks.
- As network architectures became bigger and more sophisticated, there was a growing need for automated systems for computing the necessary derivatives.
- This lecture provides an overview of these methods, called **automatic differentiation** methods.
- Before using these to differentiate loss functions w.r.t. model parameters, we describe how they can be used to take the derivative of an arbitrary function.

Chain Rule (Review)

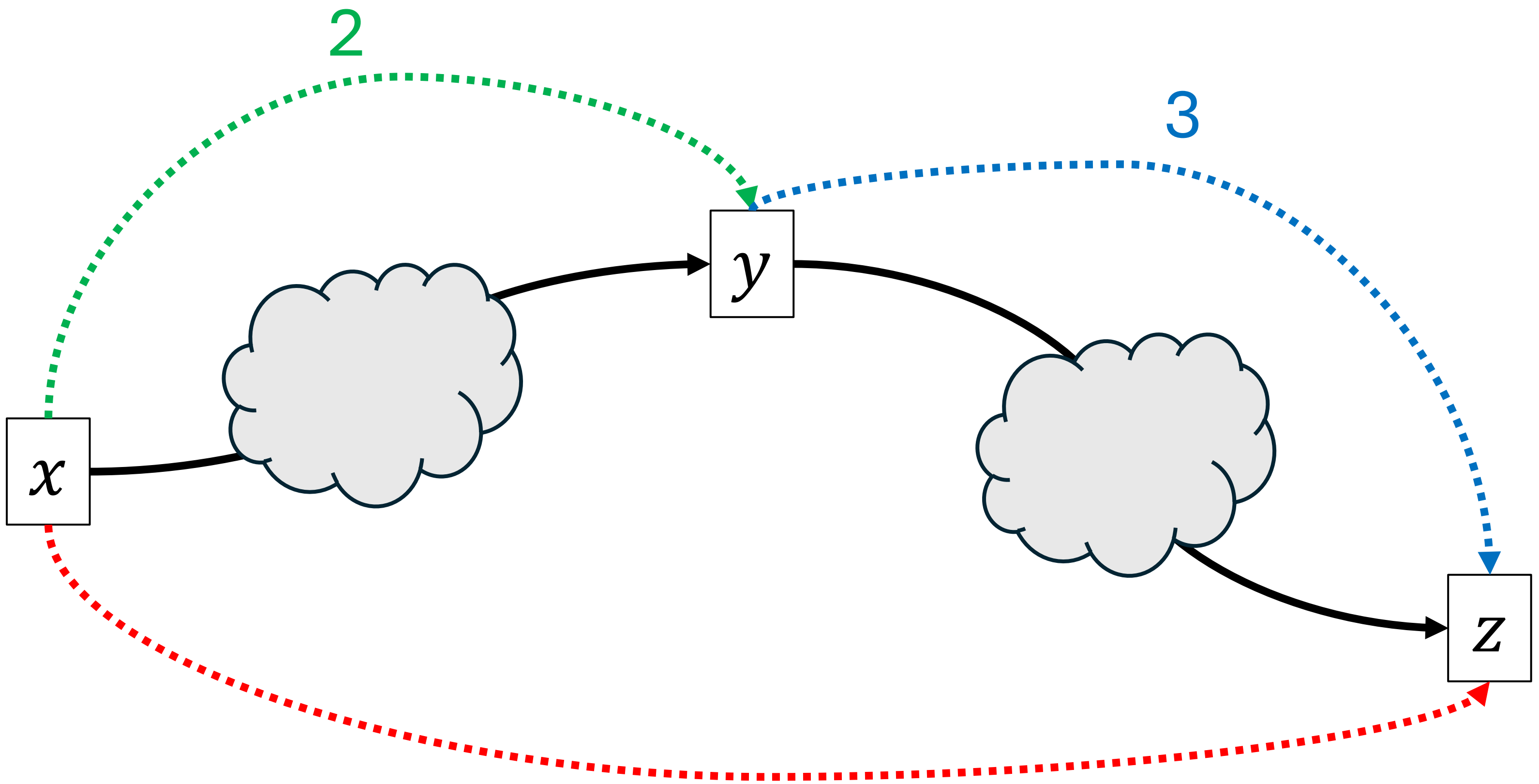
$$\frac{df\big(g(x)\big)}{dx} = \frac{df(x)}{dg(x)}\frac{dg(x)}{dx}$$

or

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

Chain Rule

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$



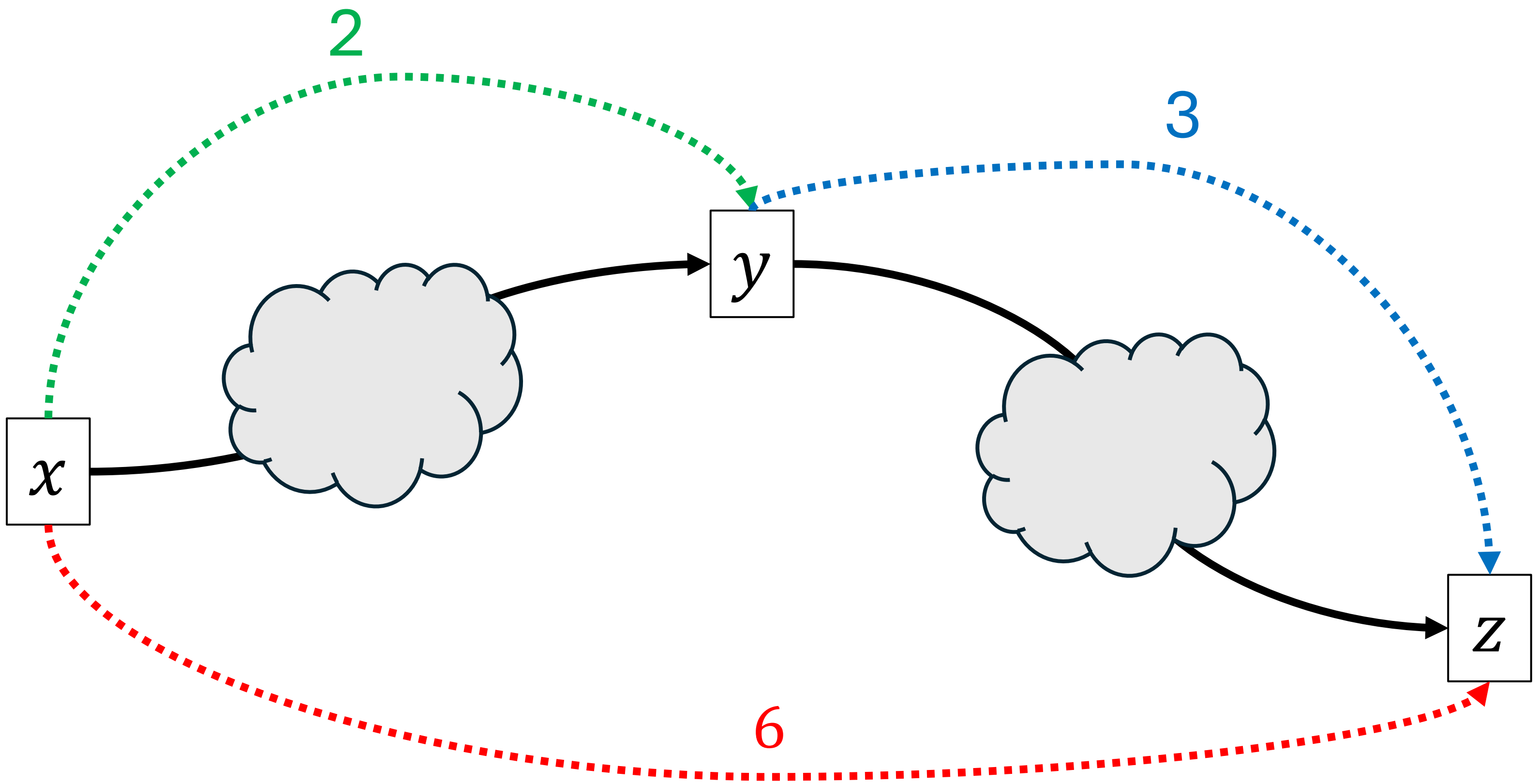
$\frac{dz}{dx}$ – How does changing x change z? =? (adding ϵ to x increases z by ? ϵ)

$\frac{dy}{dx}$ – How does changing x change y? =2 (adding ϵ to x increases y by 2ϵ)

$\frac{dz}{dy}$ – How does changing y change z? =3 (adding ϵ to y increases z by 3ϵ)

Chain Rule

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$



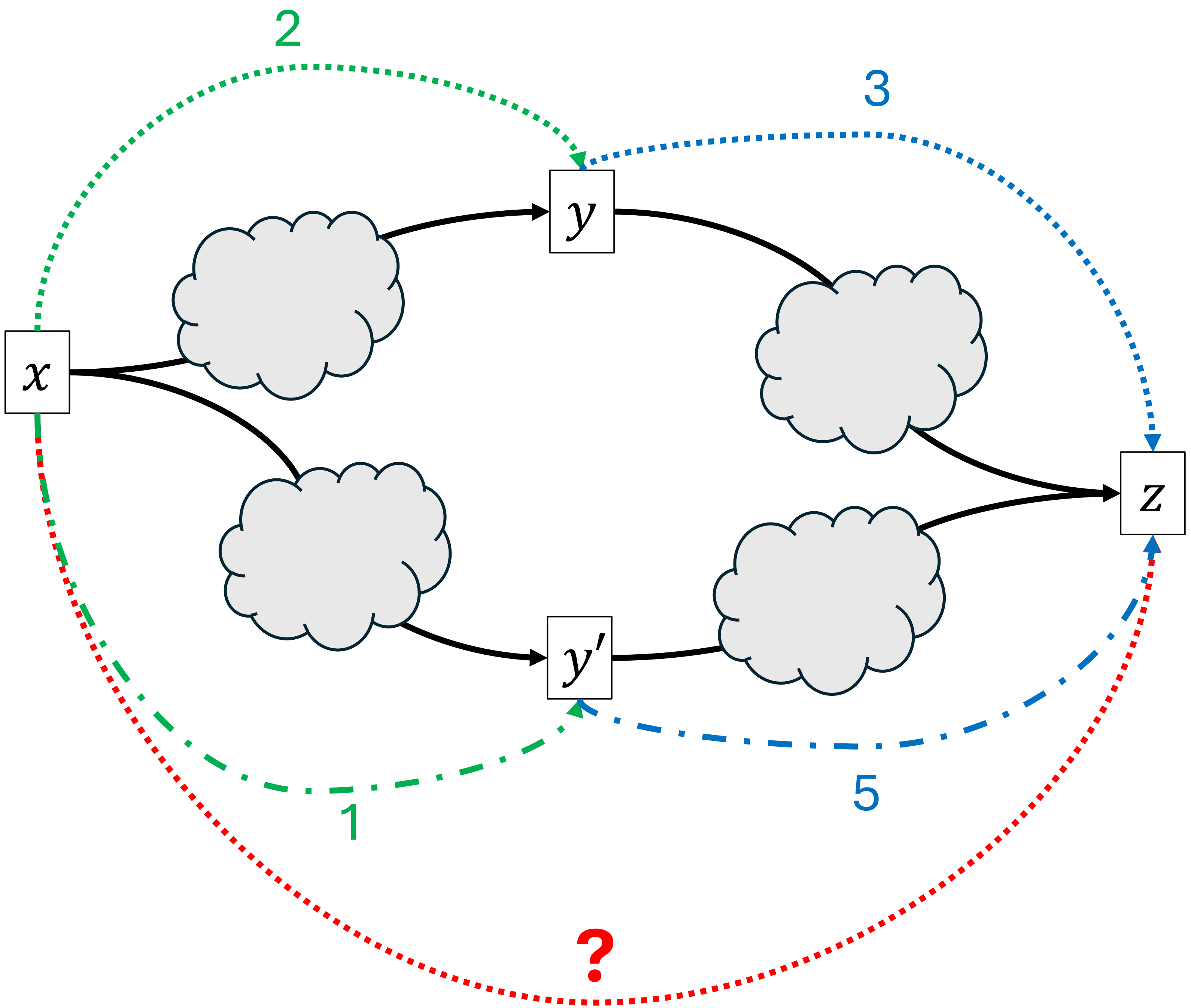
$\frac{dz}{dx}$ – How does changing x change z? =6 (adding ϵ to x increases z by 6ϵ)

$\frac{dy}{dx}$ – How does changing x change y? =2 (adding ϵ to x increases y by 2ϵ)

$\frac{dz}{dy}$ – How does changing y change z? =3 (adding ϵ to y increases z by 3ϵ)

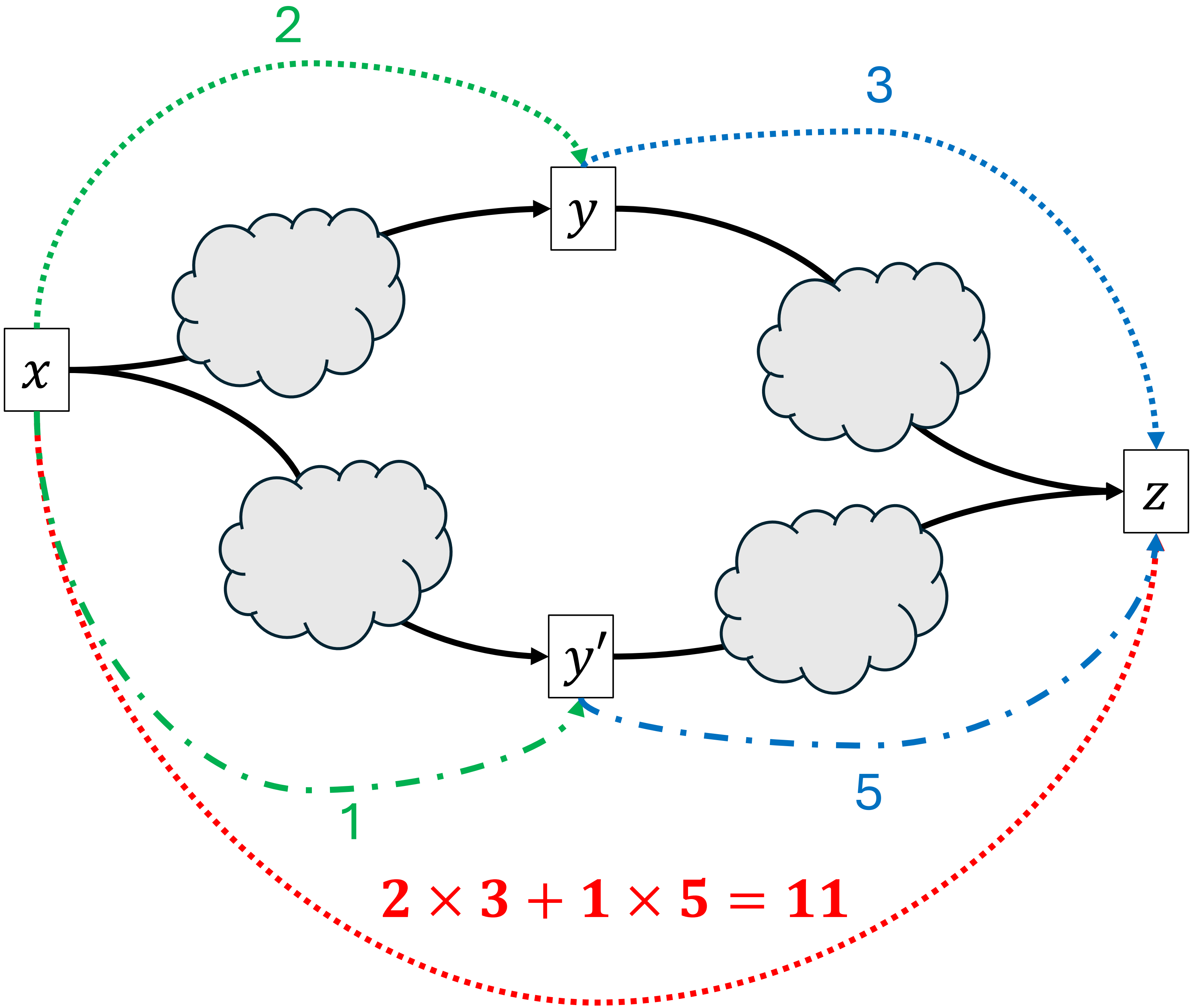
Chain Rule

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx} + \frac{dz}{dy'}\frac{dy'}{dx}$$



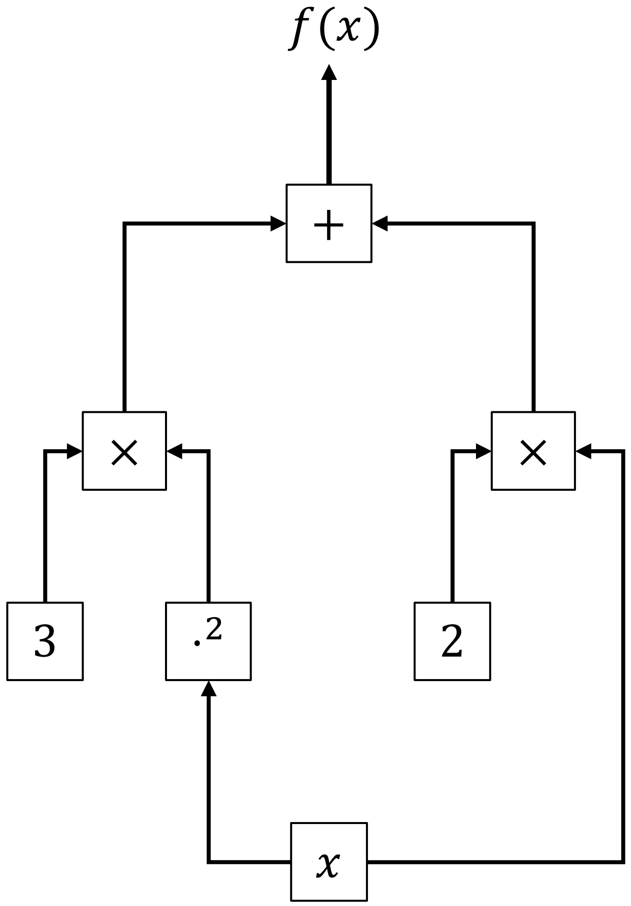
Chain Rule

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx} + \frac{dz}{dy'}\frac{dy'}{dx}$$



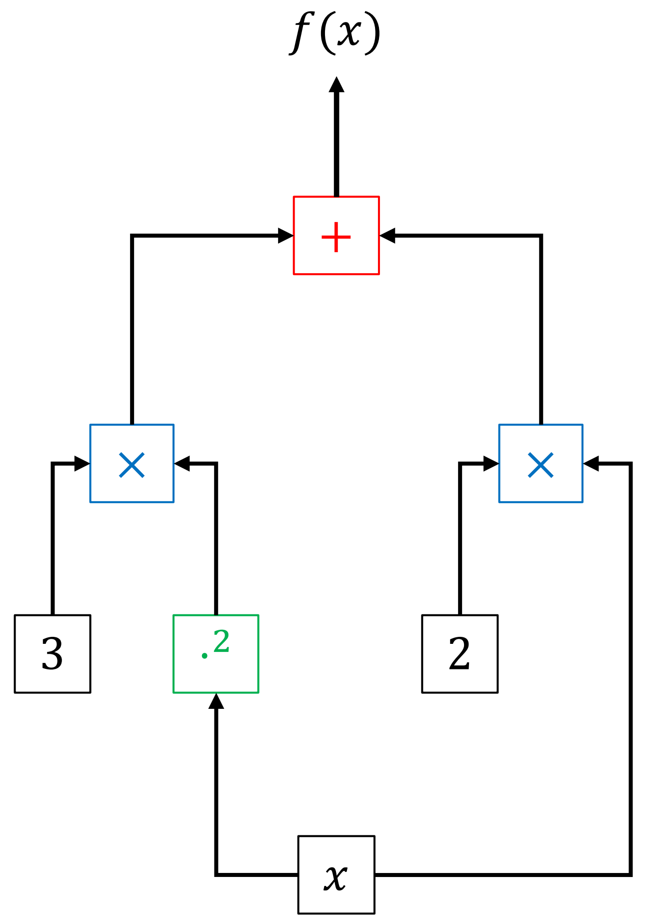
Expression Trees

- Math expressions like function definitions can be converted into *expression trees*.
  - Each internal node is a math operator.
  - Each leaf node is a constant or variable.
- Example: $f(x) = 3x^2 + 2x$



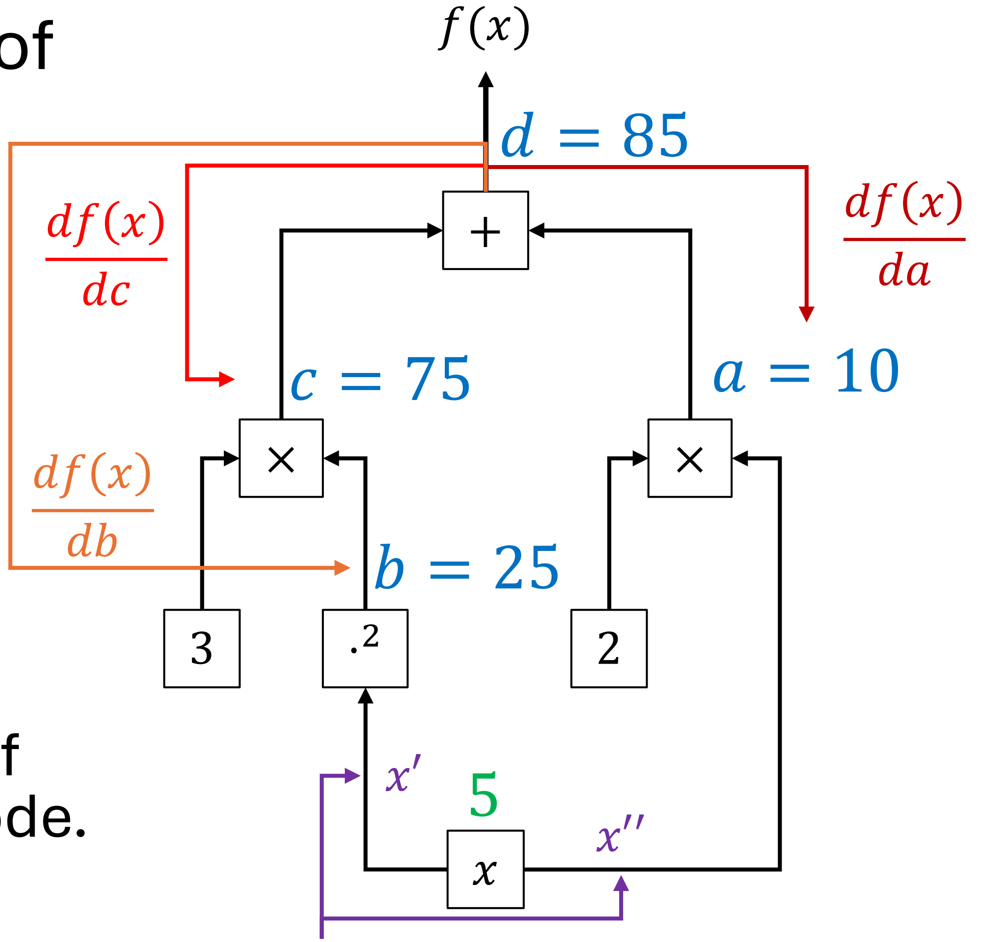
$$f(x) = 3x^2 + 2x$$

- Each math operator (internal node) can be viewed as a function.
- We can view this expression as the composition of many functions:
  - $f_1(x) = x^2$
  - $f_2(x, y) = xy$
  - $f_3(x, y) = x + y$
  - $f(x) = f_3\left(f_2(3, f_1(x)), f_2(2, x)\right)$
- We can apply the chain rule to break the derivative, $\frac{df(x)}{dx}$, into many smaller problems!



Automatic Differentiation

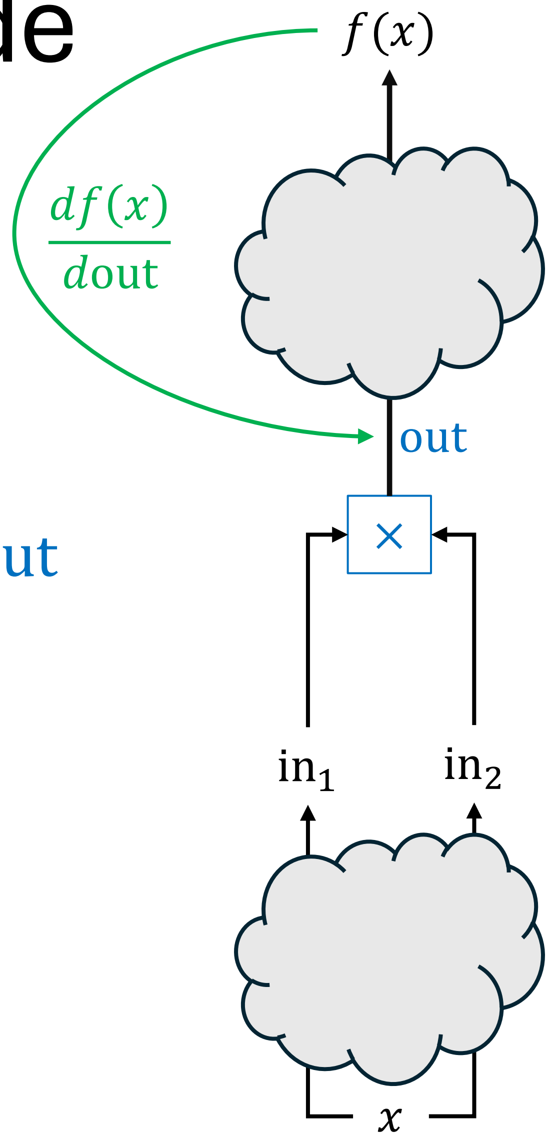
- **Goal**: Compute $\frac{df(x)}{dx}$, for some value of x
  - Example: $x = 5$
- Step 1: Run a "forwards pass"
  - Evaluate the expression tree, computing values from the bottom to the top.
- Step 2: Run a "backwards pass"
  - Loop over nodes from the top to the bottom.
  - For each node, compute the derivative of $f(x)$ with respect to each *input* of the node.



We write x' and x'' so that we can talk about the two paths, $\frac{df(x)}{dx'}$ and $\frac{df(x)}{dx''}$

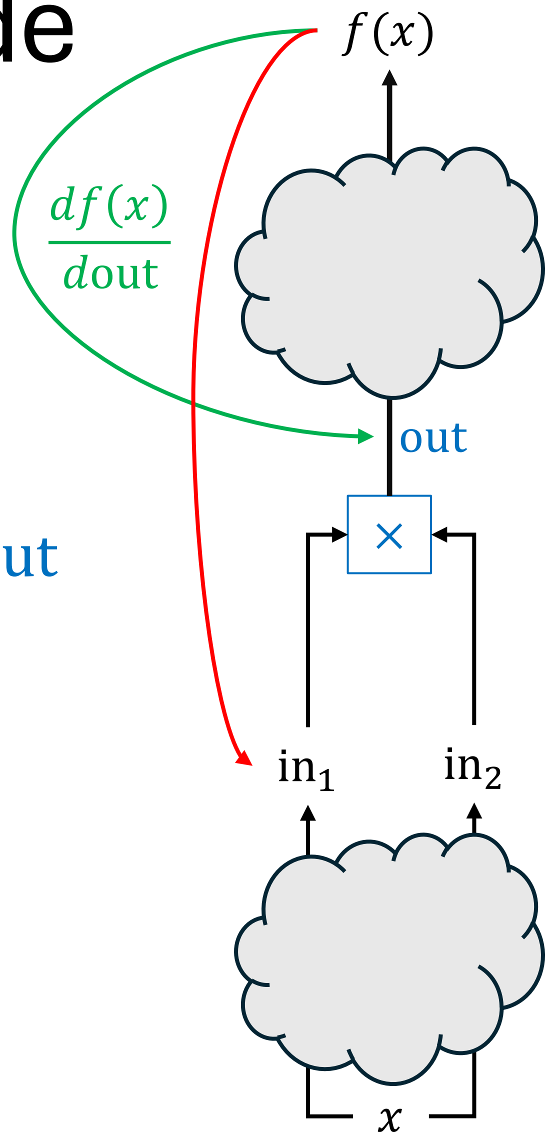
Backwards Pass: Multiplication Node

- We want to compute $\partial f(x)/\partial \mathrm{in}_1$ and $\partial f(x)/\partial \mathrm{in}_2$
- Assume that we know:
  - The value of the inputs: in_1 and in_2
    - These were computed during the forwards pass
  - The derivative of $f(x)$ *with respect to* (w.r.t.) the output out of the multiplication function, $\times$.
    - This is $\frac{df(x)}{d\mathrm{out}}$
    - This was computed earlier in the backwards pass by the node "above" the multiplication node.



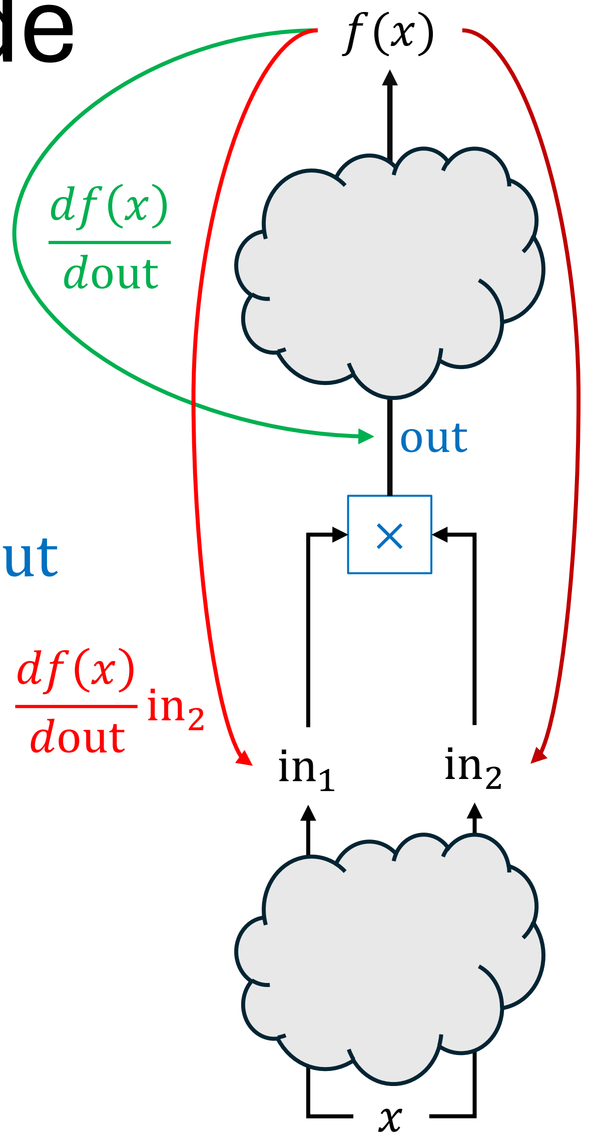
Backwards Pass: Multiplication Node

- We want to compute $\partial f(x)/\partial \mathrm{in}_1$ and $\partial f(x)/\partial \mathrm{in}_2$
- Assume that we know:
  - The value of the inputs: in_1 and in_2
    - These were computed during the forwards pass
  - The derivative of $f(x)$ *with respect to* (w.r.t.) the output out of the multiplication function, $\times$.
    - This is $\frac{df(x)}{d\mathrm{out}}$
    - This was computed earlier in the backwards pass by the node "above" the multiplication node.
- $\frac{df(x)}{d\mathrm{in}_1} = \frac{df(x)}{d\mathrm{out}} \frac{d\mathrm{out}}{d\mathrm{in}_1} = ?$



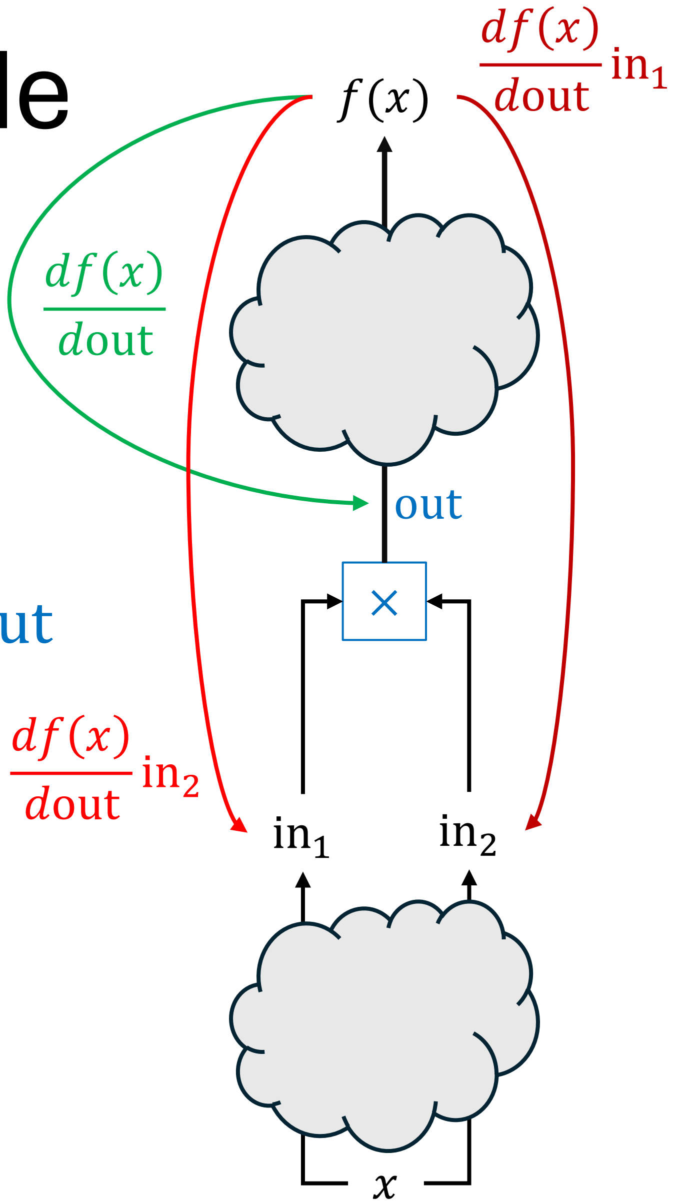
Backwards Pass: Multiplication Node

- We want to compute $\partial f(x)/\partial \text{in}_1$ and $\partial f(x)/\partial \text{in}_2$
- Assume that we know:
  - The value of the inputs: in_1 and in_2
    - These were computed during the forwards pass
  - The derivative of $f(x)$ *with respect to* (w.r.t.) the output out of the multiplication function, $\times$.
    - This is $\frac{df(x)}{d\text{out}}$
    - This was computed earlier in the backwards pass by the node "above" the multiplication node.
- $\frac{df(x)}{d\text{in}_1} = \frac{df(x)}{d\text{out}} \frac{d\text{out}}{d\text{in}_1} = \frac{df(x)}{d\text{out}} \text{in}_2$
- $\frac{df(x)}{d\text{in}_1} = \frac{df(x)}{d\text{out}} \frac{d\text{out}}{d\text{in}_2} = ?$



Backwards Pass: Multiplication Node

- We want to compute $\partial f(x)/\partial \mathrm{in}_1$ and $\partial f(x)/\partial \mathrm{in}_2$
- Assume that we know:
  - The value of the inputs: in_1 and in_2
    - These were computed during the forwards pass
  - The derivative of $f(x)$ *with respect to* (w.r.t.) the output out of the multiplication function, $\times$.
    - This is $\frac{df(x)}{d\mathrm{out}}$
    - This was computed earlier in the backwards pass by the node "above" the multiplication node.
- $\frac{df(x)}{d\mathrm{in}_1} = \frac{df(x)}{d\mathrm{out}}\frac{d\mathrm{out}}{d\mathrm{in}_1} = \frac{df(x)}{d\mathrm{out}}\mathrm{in}_2$
- $\frac{df(x)}{d\mathrm{in}_1} = \frac{df(x)}{d\mathrm{out}}\frac{d\mathrm{out}}{d\mathrm{in}_2} = \frac{df(x)}{d\mathrm{out}}\mathrm{in}_1$



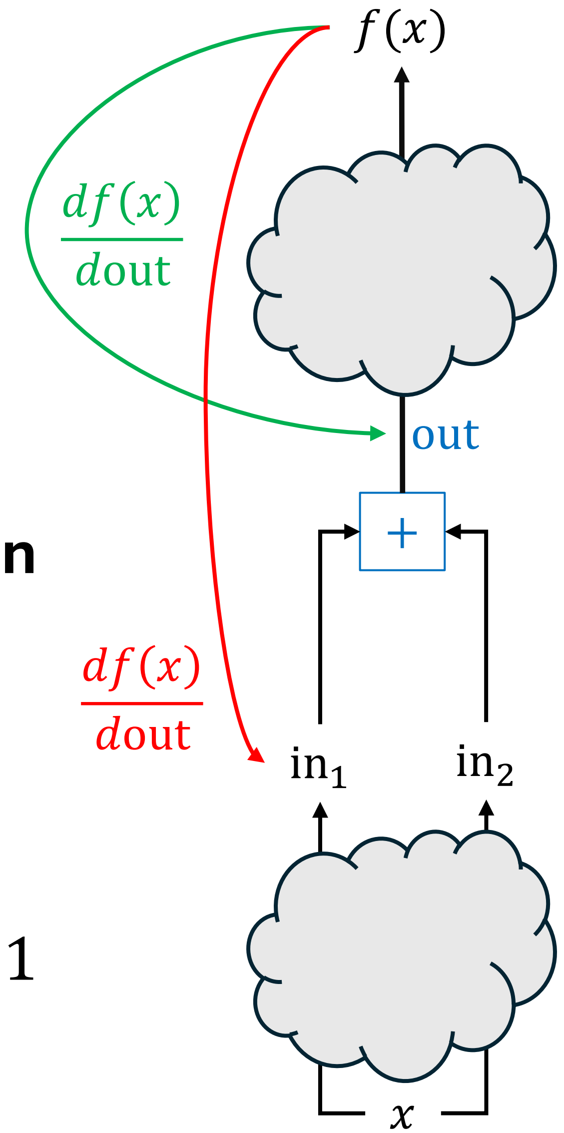
Backwards Pass

- For each math operator ($+, -, \times, \frac{a}{b}, \cdot^2$, …) used by a parametric model, derive the expression for the derivative of $f(x)$ with respect to each input of the operator, assuming:
  - The values of all inputs to the operator are known
    - They will be computed during the forwards pass.
  - The derivative of $f(x)$ w.r.t. the output of the operator is known
    - It will already have been computed in the backwards pass.

Backwards Pass: **Addition** Node

- We want to compute $\partial f(x)/\partial \mathrm{in}_1$ and $\partial f(x)/\partial \mathrm{in}_2$
- Assume that we know:
  - The value of the inputs: in_1 and in_2
    - These were computed during the forwards pass
  - The derivative of $f(x)$ w.r.t. the output out of the **addition** function, $+$.
    - This is $\frac{df(x)}{d\mathrm{out}}$
    - This was computed earlier in the backwards pass by the node "above" the multiplication node.
- $\frac{df(x)}{d\mathrm{in}_1} = \frac{df(x)}{d\mathrm{out}} \frac{d\mathrm{out}}{d\mathrm{in}_1} = \frac{df(x)}{d\mathrm{out}}$

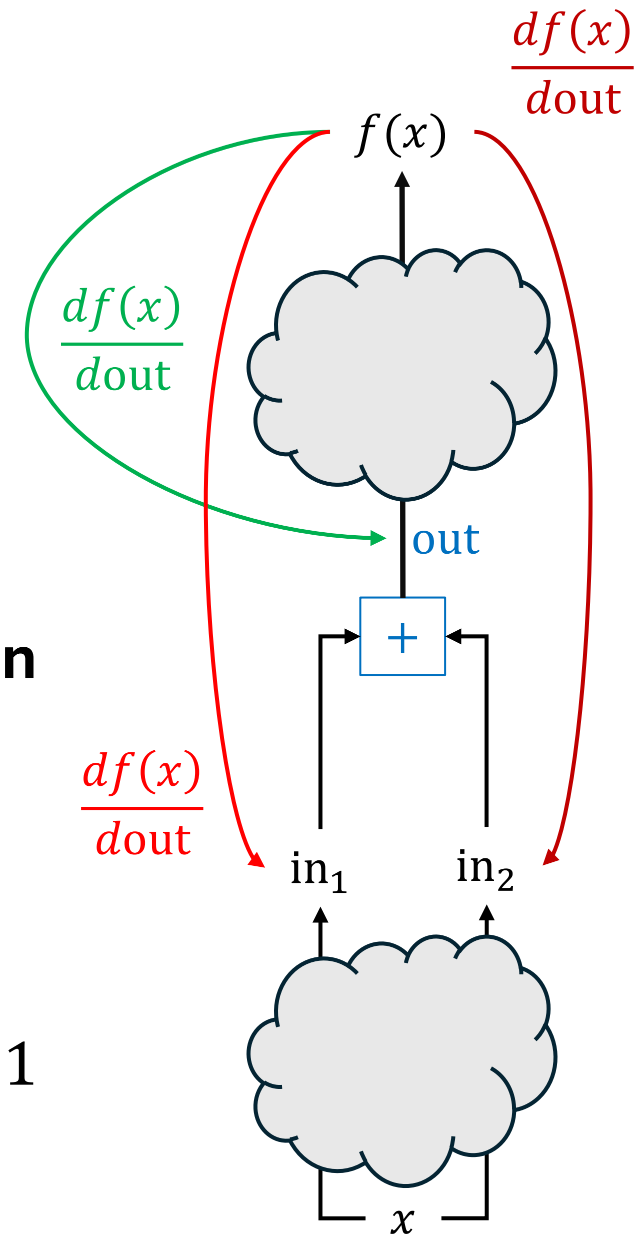
$$\frac{d\mathrm{out}}{d\mathrm{in}_1} = 1$$



Backwards Pass: **Addition** Node

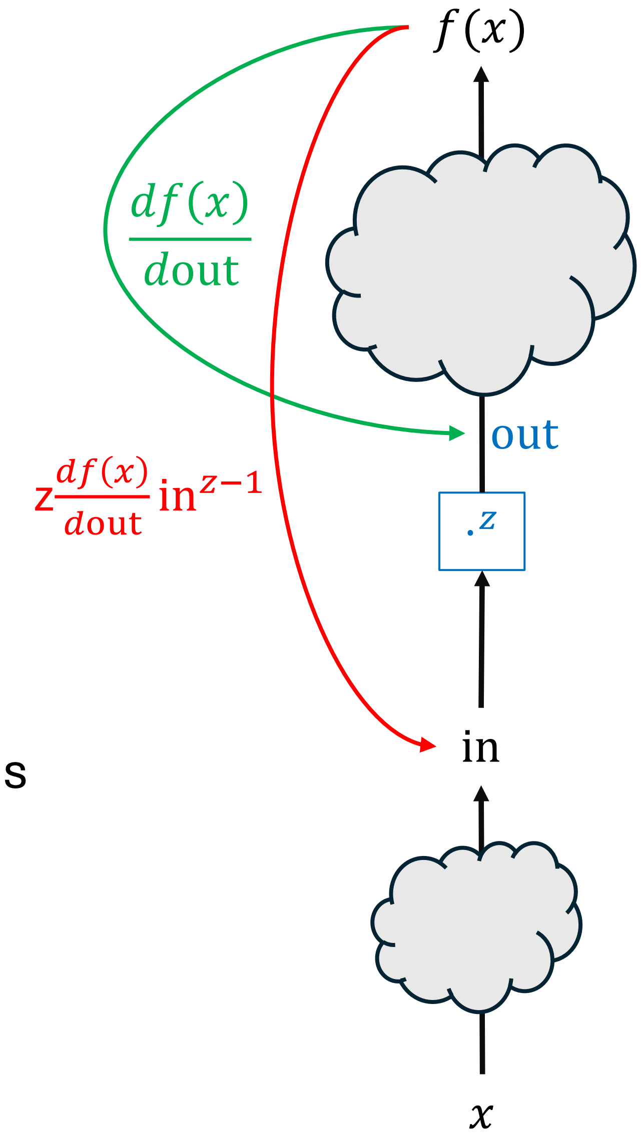
- We want to compute $\partial f(x)/\partial \mathrm{in}_1$ and $\partial f(x)/\partial \mathrm{in}_2$
- Assume that we know:
  - The value of the inputs: in_1 and in_2
    - These were computed during the forwards pass
  - The derivative of $f(x)$ w.r.t. the output out of the **addition** function, $+$.
    - This is $\frac{df(x)}{d\mathrm{out}}$
    - This was computed earlier in the backwards pass by the node "above" the multiplication node.
- $\frac{df(x)}{d\mathrm{in}_1} = \frac{df(x)}{d\mathrm{out}} \frac{d\mathrm{out}}{d\mathrm{in}_1} = \frac{df(x)}{d\mathrm{out}}$
- $\frac{df(x)}{d\mathrm{in}_1} = \frac{df(x)}{d\mathrm{out}} \frac{d\mathrm{out}}{d\mathrm{in}_2} = \frac{df(x)}{d\mathrm{out}}$

$$\frac{d\mathrm{out}}{d\mathrm{in}_1} = 1$$



Backwards Pass: **Exponent** Node

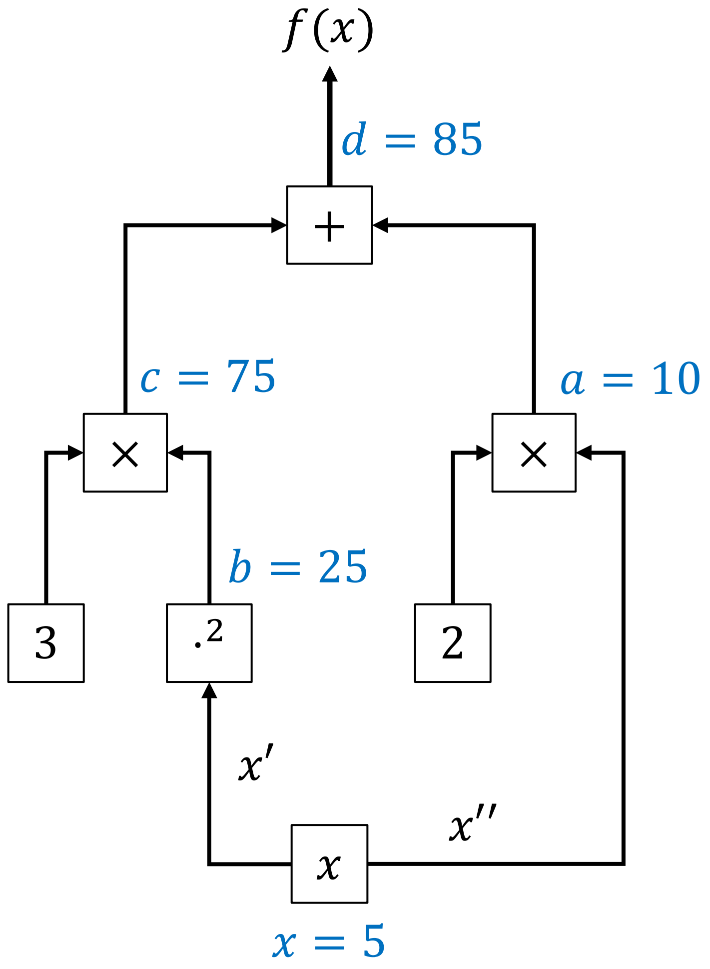
- We want to compute $\partial f(x)/\partial \text{in}$.
- Assume z is a constant.
- Assume that we know:
  - The value of the input in from the forwards pass
  - The derivative of $f(x)$ w.r.t. the output out of the **exponentiation** function, $(\cdot)^z$.
    - This is $\frac{df(x)}{d\text{out}}$, as was computed previously in the backwards pass



- $\frac{df(x)}{d\text{in}} = \frac{df(x)}{d\text{out}} \frac{d\text{out}}{d\text{in}} = \frac{df(x)}{d\text{out}} \times z \times \text{in}^{z-1}$

Compute $\frac{\mathrm{d}f}{\mathrm{d}x}$ for $f(x) = 3x^2 + 2x$ at $x = 5$

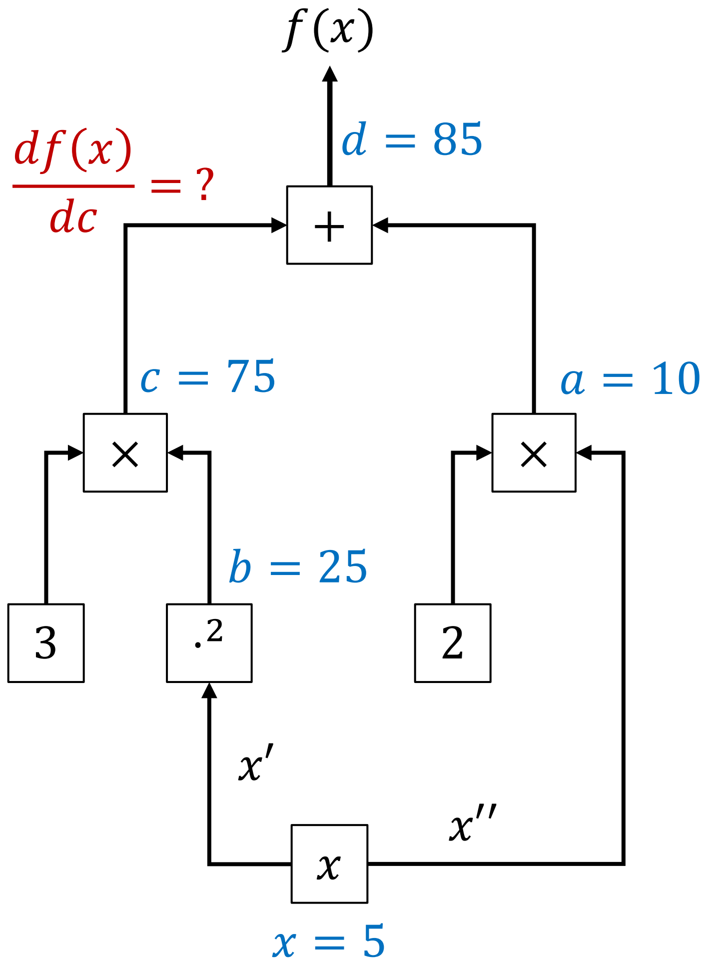
Forwards Pass



Compute $\frac{df}{dx}$ for $f(x) = 3x^2 + 2x$ at $x = 5$

Forwards Pass

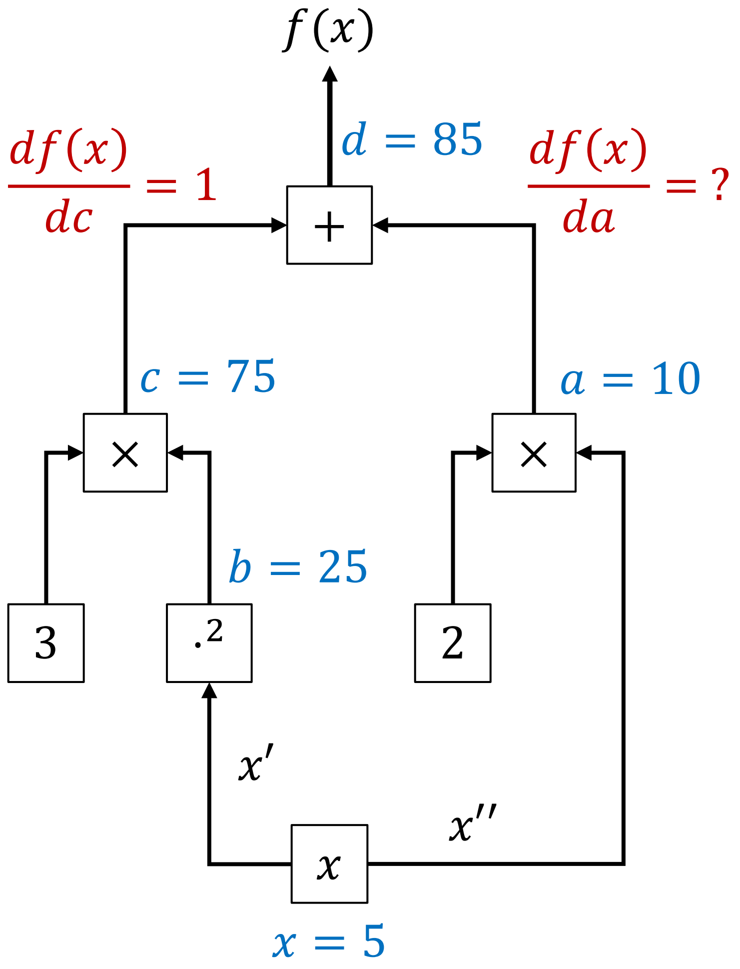
Backwards Pass



Compute $\frac{\mathrm{d}f}{\mathrm{d}x}$ for $f(x) = 3x^2 + 2x$ at $x = 5$

Forwards Pass

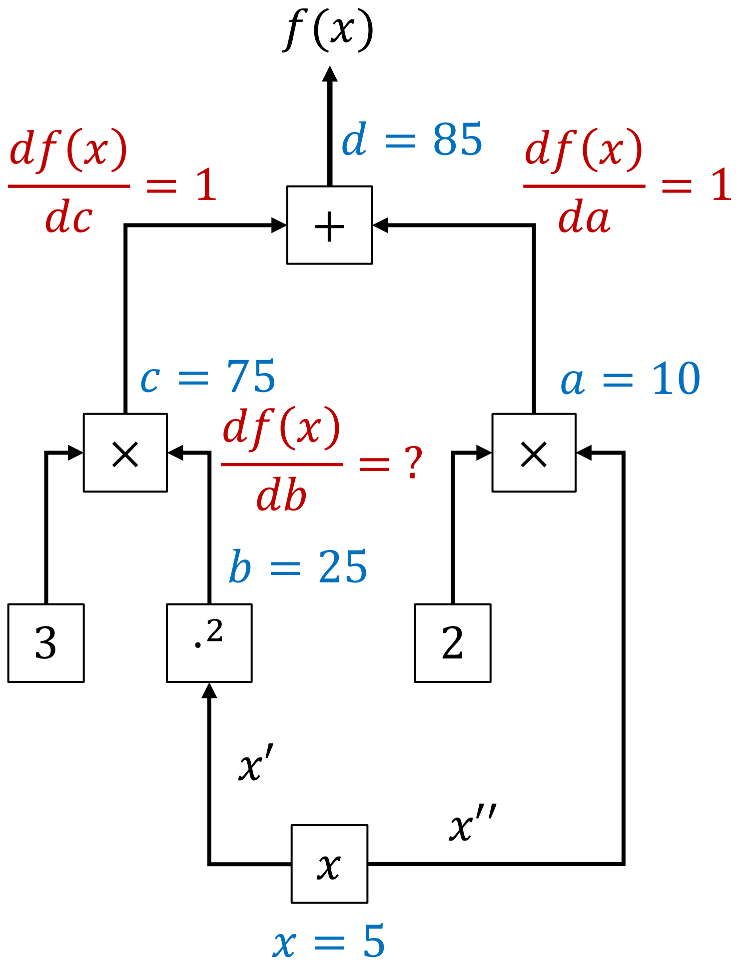
Backwards Pass



Compute $\frac{\mathrm{d}f}{\mathrm{d}x}$ for $f(x) = 3x^2 + 2x$ at $x = 5$

Forwards Pass

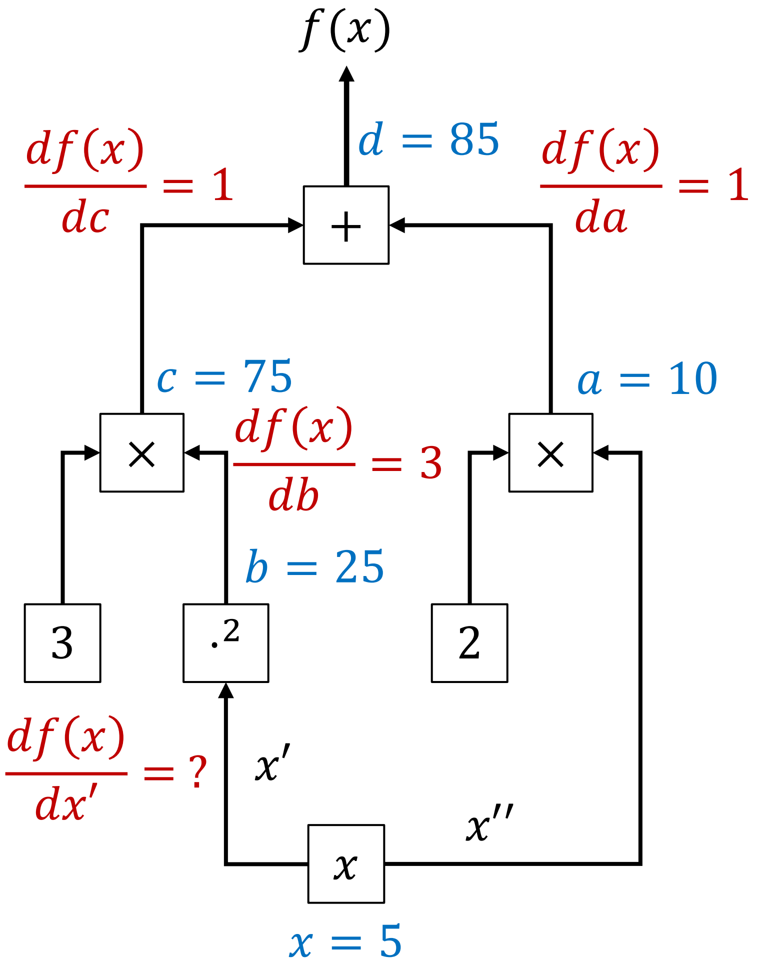
Backwards Pass



Compute $\frac{\mathrm{d}f}{\mathrm{d}x}$ for $f(x) = 3x^2 + 2x$ at $x = 5$

Forwards Pass

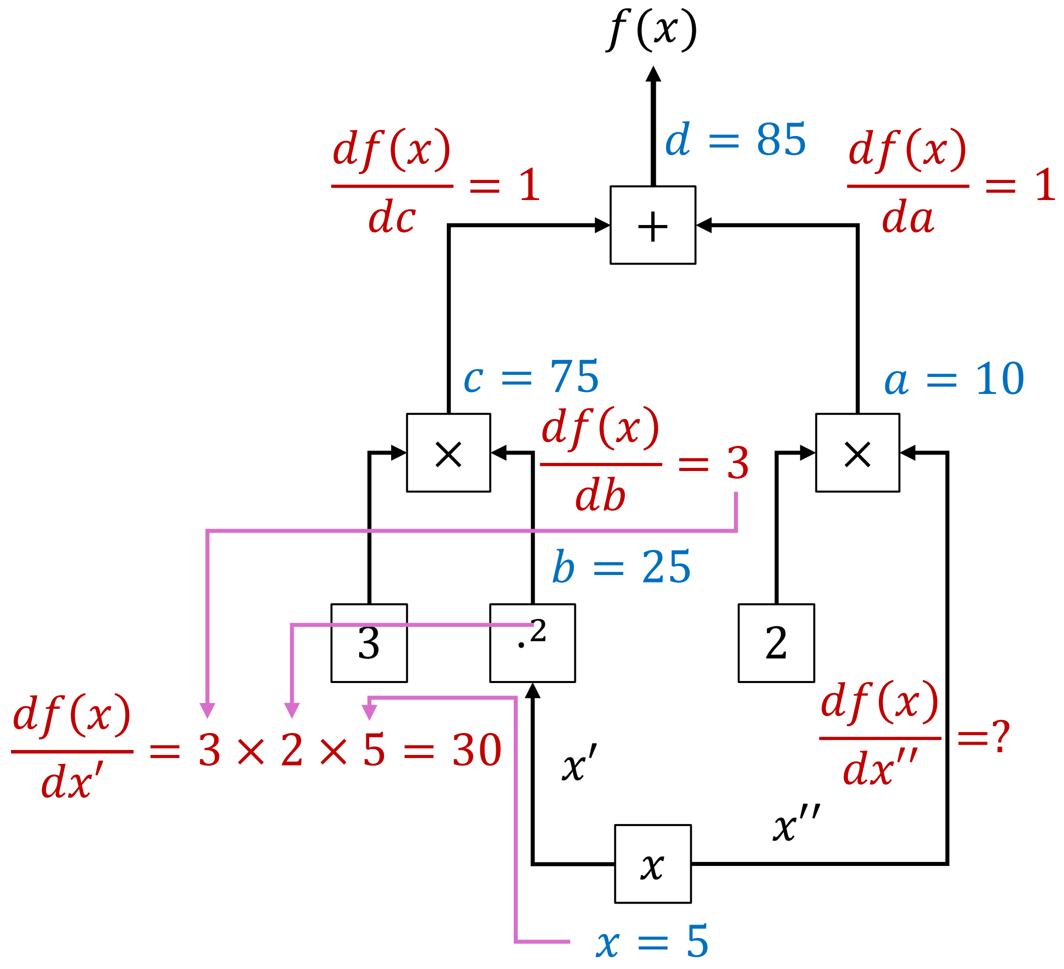
Backwards Pass



Compute $\frac{\mathrm{d}f}{\mathrm{d}x}$ for $f(x) = 3x^2 + 2x$ at $x = 5$

Forwards Pass

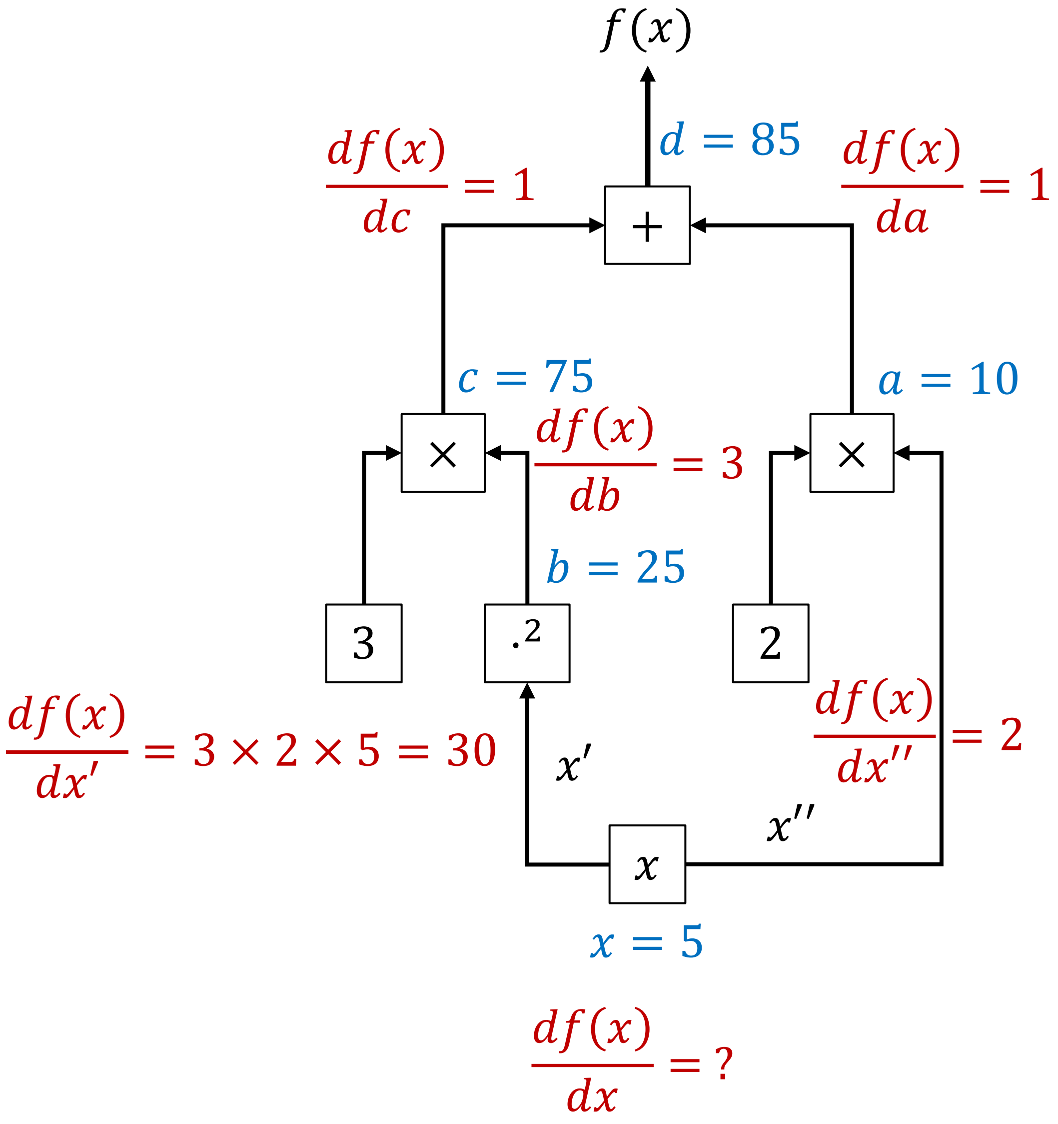
Backwards Pass



Compute $\frac{\mathrm{d}f}{\mathrm{d}x}$ for $f(x) = 3x^2 + 2x$ at $x = 5$

Forwards Pass

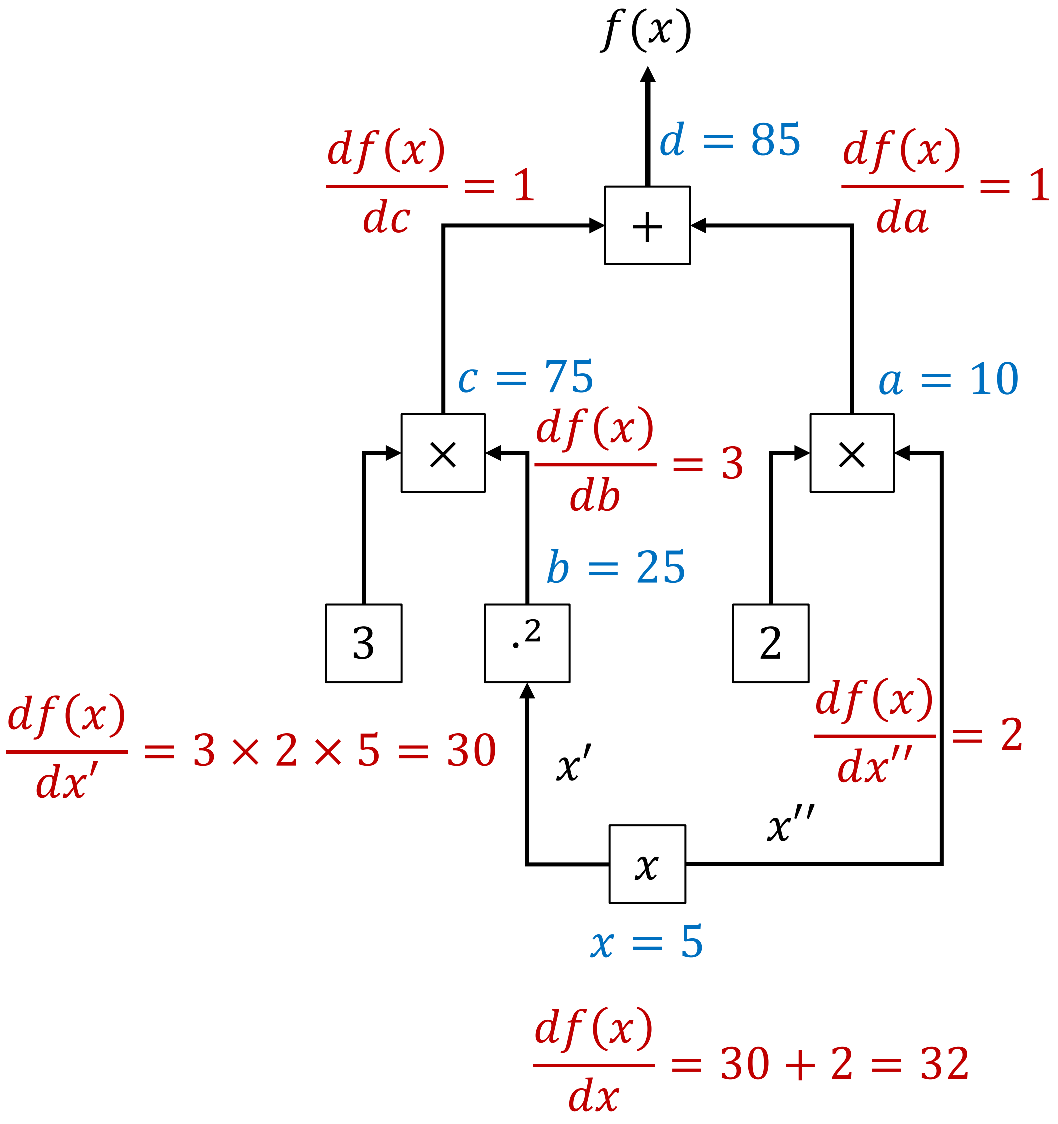
Backwards Pass



Compute $\frac{\mathrm{d}f}{\mathrm{d}x}$ for $f(x) = 3x^2 + 2x$ at $x = 5$

Forwards Pass

Backwards Pass



Automatic Differentiation

- **Automatic differentiation** tools take functions as input
  - Typically these functions are implemented as code, e.g., *python functions*.
- They can then be used to take the derivative of the function with respect to the arguments (inputs).
- There are several methods for automatic differentiation, with different pros and cons.
  - **Forwards Mode Automatic Differentiation**: Runs one forwards pass (no backwards pass!). Computes the derivative of the output w.r.t. a *single* scalar input.
  - **Reverse Mode Automatic Differentiation**: The strategy we have described.
    - Requires a forward and backwards pass.
    - Can compute the derivative with respect to all inputs with one forwards+backwards pass.
    - This is most common for automatically differentiating ML models and loss functions.
  - Others include **symbolic differentiation** (manipulating the mathematical expressions to calculate expressions for the derivative) and **finite difference methods** (beyond the scope of this course).

The remainder of this presentation covers:

`10.1 Automatic Differentiation for Functions.ipynb`

Python Autograd

- Autograd is a tool for differentiating functions defined by Python code.
- Autograd provides the function `grad`, which uses reverse mode automatic differentiation.
- Installation:

```
pip install autograd
```

- Import:

```python
from autograd import grad
```

Autograd

- Weight vectors are usually represented as `ndarray` objects from `numpy`.
- Autograd provides a wrapper for numpy that enables automatic differentiation with numpy objects.

```
import autograd.numpy as np
```

Autograd Basic Usage

- Define a function that you would like to differentiate:

```
def f(x):
    return 3 * (x**2) + (2 * x)
```

- Call the `grad` function to get a new function that returns the gradient (derivative)

```
f_prime = grad(f)
```

- Evaluate the `f_prime` function to get the derivative for a value of x

```
display(f"The derivative is: {f_prime(5.0)}.")
```

```
'The derivative is: 32.0.'
```

Autograd (Multiple Inputs)

- The second argument or grad specifies the input to take the derivative with respect to (default = 0)

```
def f(x, y):
    return 3 * x**2 + 2 * y - 7
```

$$f(x, y) = 3x^2 + 2y - 7$$

Autograd (Multiple Inputs)

- The second argument or grad specifies the input to take the derivative with respect to (default = 0)

```
def f(x, y):
    return 3 * x**2 + 2 * y - 7

partial_x = grad(f, 0)  # Partial derivative with respect to x. This is equivalent to grad(f).
partial_y = grad(f, 1)  # Partial derivative with respect to y
```

$$f(x, y) = 3x^2 + 2y - 7$$

Autograd (Multiple Inputs)

- The second argument or grad specifies the input to take the derivative with respect to (default = 0)

```python
def f(x, y):
    return 3 * x**2 + 2 * y - 7

partial_x = grad(f, 0)  # Partial derivative with respect to x. This is equivalent to grad(f).
partial_y = grad(f, 1)  # Partial derivative with respect to y

display(f"The partial derivative w.r.t. x is: {partial_x(3.0, 5.0)}.")
display(f"The partial derivative w.r.t. y is: {partial_y(3.0, 5.0)}.")
```

$$f(x, y) = 3x^2 + 2y - 7$$

Autograd (Multiple Inputs)

- The second argument or grad specifies the input to take the derivative with respect to (default = 0)

```
def f(x, y):
    return 3 * x**2 + 2 * y - 7

partial_x = grad(f, 0)  # Partial derivative with respect to x. This is equivalent to grad(f).
partial_y = grad(f, 1)  # Partial derivative with respect to y

display(f"The partial derivative w.r.t. x is: {partial_x(3.0, 5.0)}.")
display(f"The partial derivative w.r.t. y is: {partial_y(3.0, 5.0)}.")
```

$$f(x, y) = 3x^2 + 2y - 7$$

```
'The partial derivative w.r.t. x is: 18.0.'

'The partial derivative w.r.t. y is: 2.0.'
```

Autograd (Vector Inputs)

- Autograd can take the derivative with respect to a vector of inputs.

```
# The same function, but taking a numpy array as input
def f(inputs):
    x, y = inputs
    return 3 * x**2 + 2 * y - 7

# Now, the gradient function returns the gradient with respect to the entire numpy array of inputs
grad_f = grad(f)

input = np.array([3.0, 5.0])    # Create the input for which we want the derivatives w.r.t.
gradient = grad_f(input)        # Get the derivatives (the gradient)
display(f"The gradient at {input} is {gradient}")
```

$$f(x, y) = 3x^2 + 2y - 7$$

```
'The gradient at [3. 5.] is [18.  2.]'
```

End