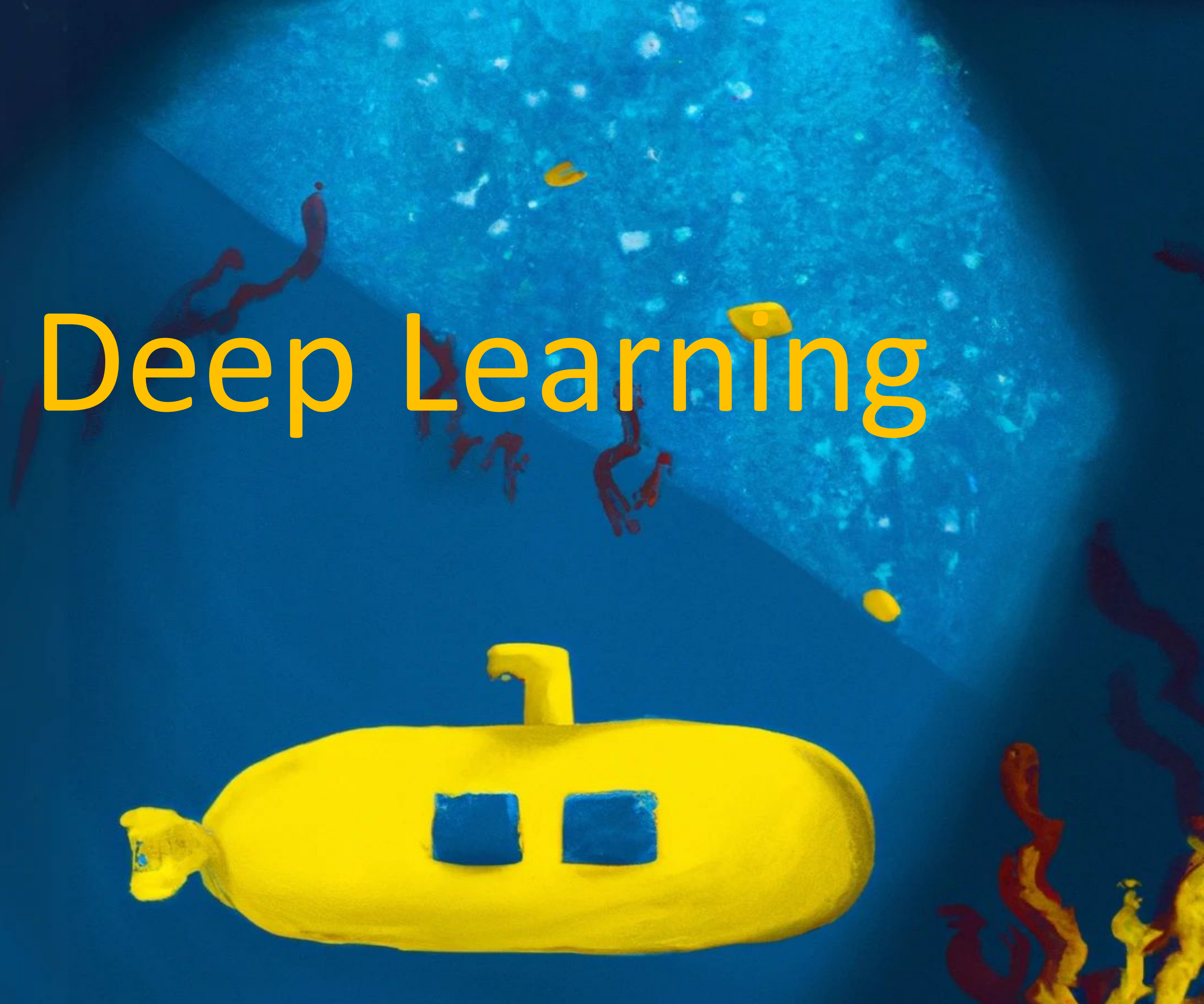
CSCI 1470/2470
Spring 2023

Ritambhara Singh

February 08, 2023
Wednesday

Deep Learning

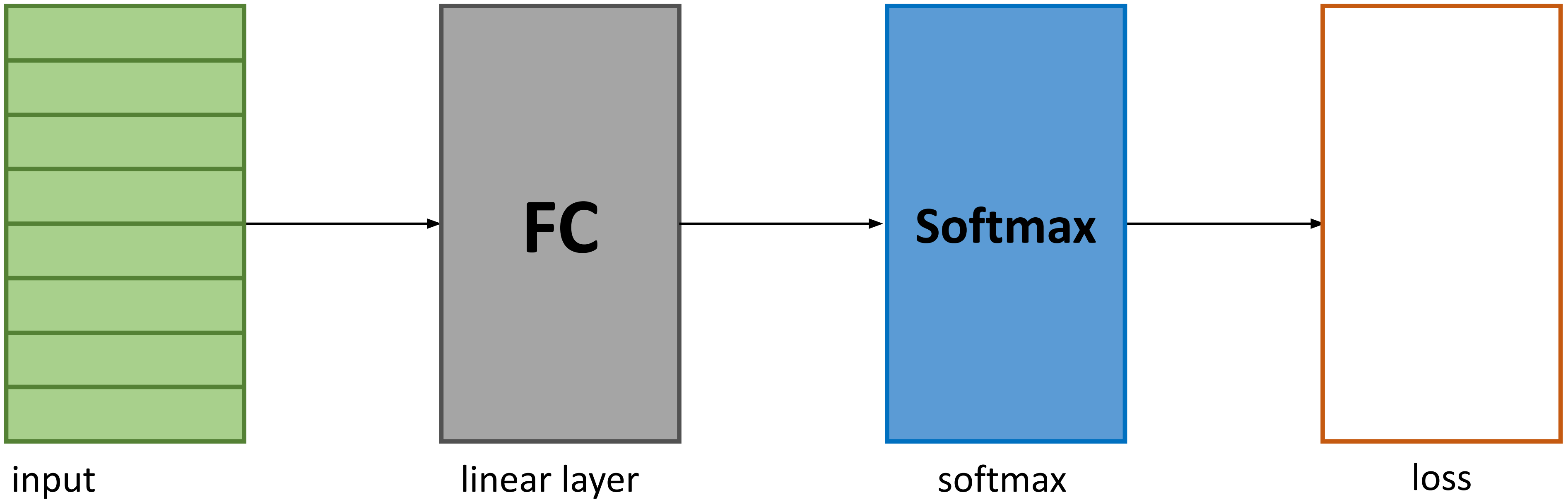


DALL-E 2 prompt “a painting of deep underwater with a yellow submarine in the bottom right corner”

Recap: Forward Pass

Compute the prediction or evaluate the loss for a single input $\boldsymbol{x}$.

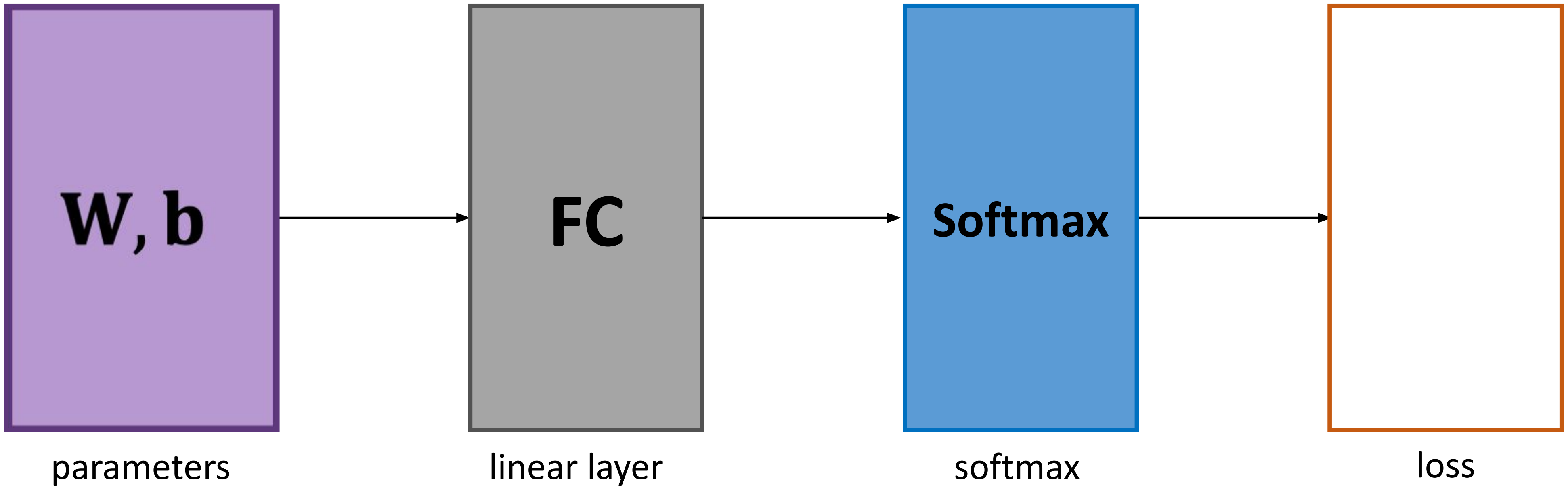
Goal of learning: Minimize the total loss for all $\boldsymbol{x}$ in training data.



Recap: Forward Pass

Compute the prediction or evaluate the loss for a single input $\boldsymbol{x}$.

Goal of learning: Minimize the total loss for all $\boldsymbol{x}$ in training data with respect to model parameters $\boldsymbol{W}, \boldsymbol{b}$.

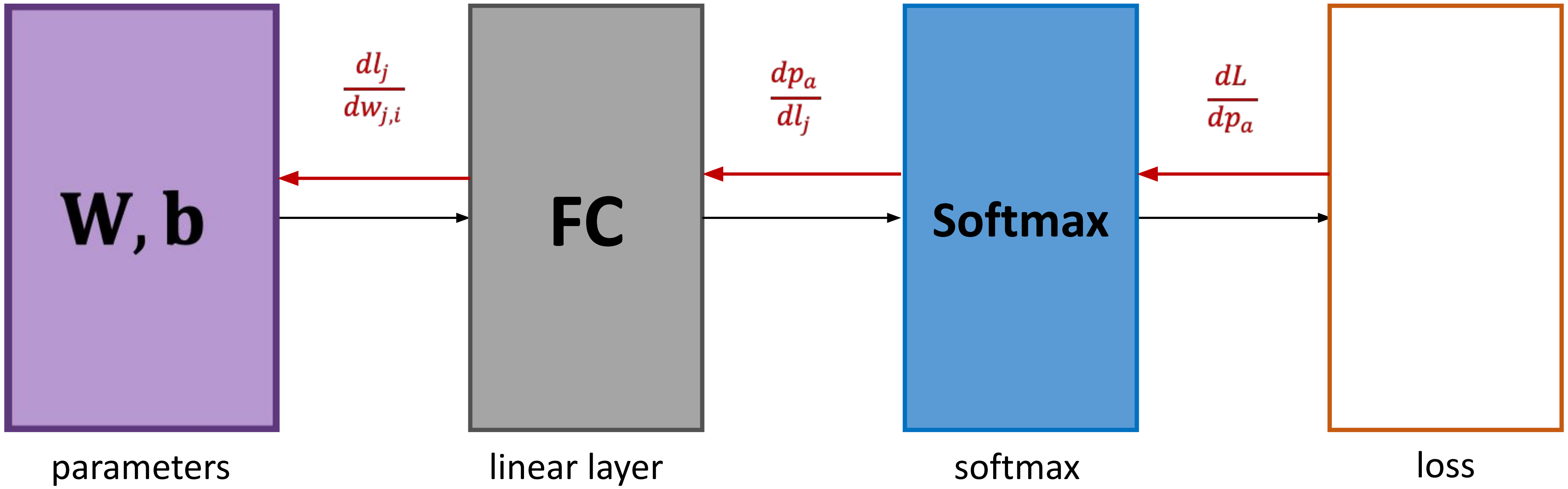


Recap: Backpropagation (Backward Pass)

Gradient descent: $\Delta W = -\alpha \, \nabla \hat{L}(W)$ and $\Delta \boldsymbol{b} = -\alpha \, \nabla \hat{L}(\boldsymbol{b})$

Backpropagation: Compute ΔW and $\Delta \boldsymbol{b}$ via chain rule.

$$\frac{dL}{dw_{j,i}} = \frac{dl_j}{dw_{j,i}} \cdot \frac{dp_a}{dl_j} \cdot \cdots$$



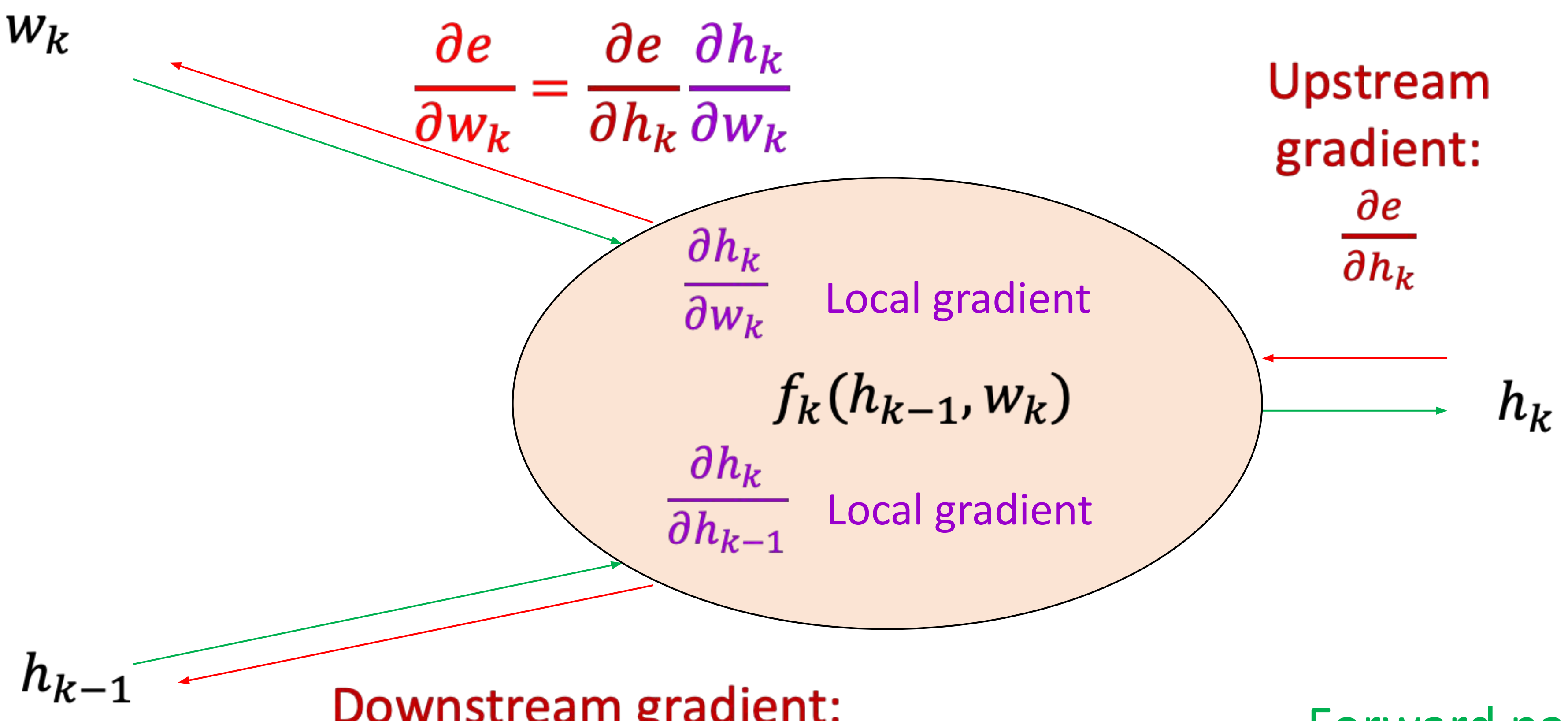
Recap: Computation graph

Parameter update:

$$\frac{\partial e}{\partial w_k} = \frac{\partial e}{\partial h_k} \frac{\partial h_k}{\partial w_k}$$

Upstream gradient:

$$\frac{\partial e}{\partial h_k}$$



Downstream gradient:

$$\frac{\partial e}{\partial h_{k-1}} = \frac{\partial e}{\partial h_k} \frac{\partial h_k}{\partial h_{k-1}}$$

Forward pass

Backward pass

Lazebnik

Today’s goal – learn about deep learning frameworks

(1) Gradient Descent pseudocode

(2) Stochastic Gradient Descent (SGD)

(3) Automatic differentiation

Putting Everything Together: Gradient Descent

 # delta_W is 2-D matrix of 0's in the shape of W

```
for each input and corresponding answer a:
  probabilities = run_network(input)
  for j in range(len(probabilities)):
    y_j = 1 if j == a else 0
    for i in range(len(input)):
      delta_W[j][i] += alpha * (y_j – probabilities[j]) * input[i]
W += delta_W
```

Forward pass

Backward pass:

Compute $\frac{\partial L}{\partial W_{ij}}$ for every W_{ij}

Over the entire dataset

Putting Everything Together: Gradient Descent

```
# delta_W is 2-D matrix of 0's in the shape of W
for each input and corresponding answer a:
  probabilities = run_network(input)
  for j in range(len(probabilities)):
    y_j = 1 if j == a else 0
    for i in range(len(input):
      delta_W[j][i] += alpha * (y_j – probabilities[j]) * input[i]
W += delta_W
```

Forward pass

Backward pass:

Compute $\frac{\partial L}{\partial W_{ij}}$ for every W_{ij}

Over the entire dataset

Gradient descent update

Gradient Descent: Limitation?

```
# delta_W is 2-D matrix of 0's in the shape of W
for each input and corresponding answer a:
  probabilities = run_network(input)
  for j in range(len(probabilities)):
    y_j = 1 if j == a else 0
    for i in range(len(input):
      delta_W[j][i] += alpha * (y_j – probabilities[j]) * input[i]
W += delta_W
```

We iterate over the ***entire*** dataset...

...to update the weights only ***once***

Stochastic Gradient Descent (SGD)

- Alternative is to train on ***batches***: small subsets of the training data
- Why *stochastic*: Each batch is **randomly** sampled from the full training data
- We update the parameters after each **batch**

Stochastic Gradient Descent: Pseudocode

```
for each batch:
  # delta_W is 2-D matrix of 0's in the shape of W
  for each input and corresponding answer a in batch:
    probabilities = run_network(input)
    for j in range(len(probabilities)):
      y_j = 1 if j == a else 0
      for i in range(len(input):
        delta_W[j][i] += alpha * (y_j – probabilities[j]) * input[i]
  W += delta_W
```

Stochastic Gradient Descent: Pseudocode

```
for each batch:
  # delta_W is 2-D matrix of 0's in the shape of W
  for each input and corresponding answer a  in batch:
    probabilities = run_network(input)
    for j in range(len(probabilities)):
      y_j = 1 if j == a else 0
      for i in range(len(input):
        delta_W[j][i] += alpha * (y_j – probabilities[j]) * input[i]
  W += delta_W
```

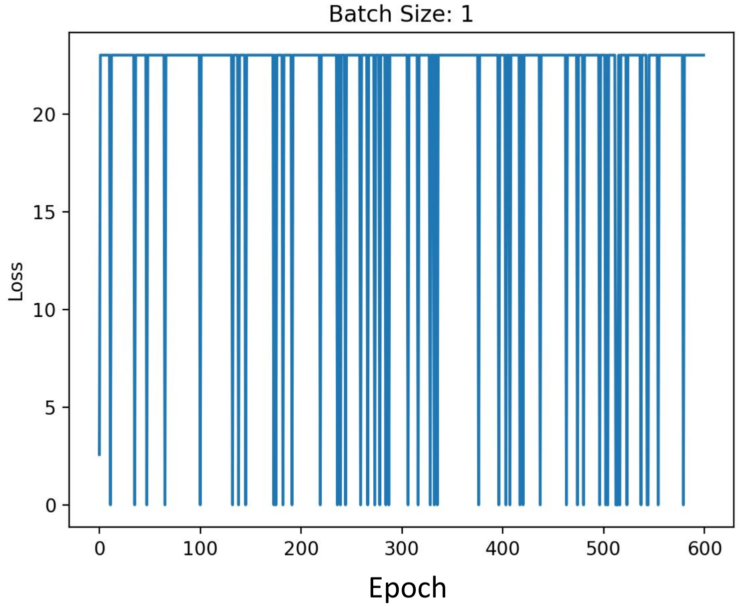
Now we update weights ***after every batch***

Stochastic Gradient Descent (SGD)

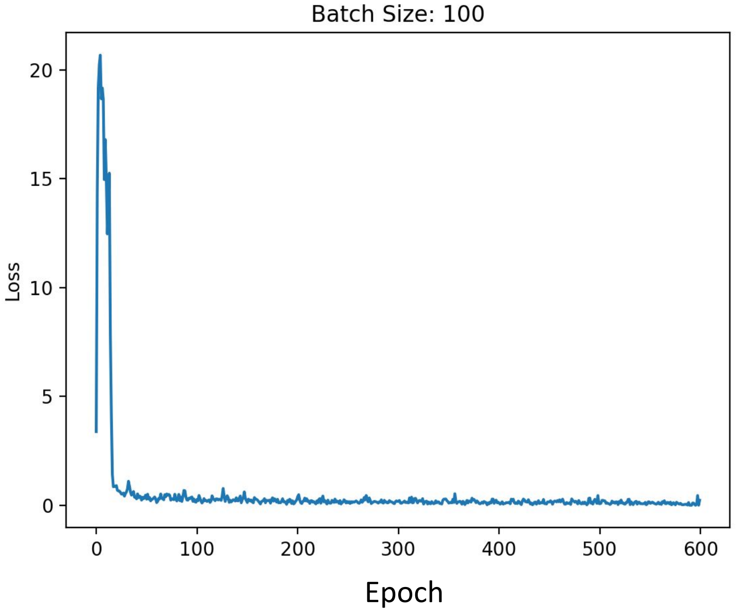
- Train on ***batches***: small subsets of the training data
- We update the parameters after each batch
- This makes the training process *stochastic* or non-deterministic: -
  *batches are a *random* subsample of the data
  ***do not provide the gradient that the entire dataset** as a whole would provide at once
- Formally: the gradient of a randomly-sampled batch is an unbiased estimator of the gradient over the whole dataset
  - “Unbiased”: expected value == the true gradient, but may have large variance (i.e. the gradient may ‘jitter around’ a lot)

What size should the batch be?

Small batch size:
Fast, jittery updates

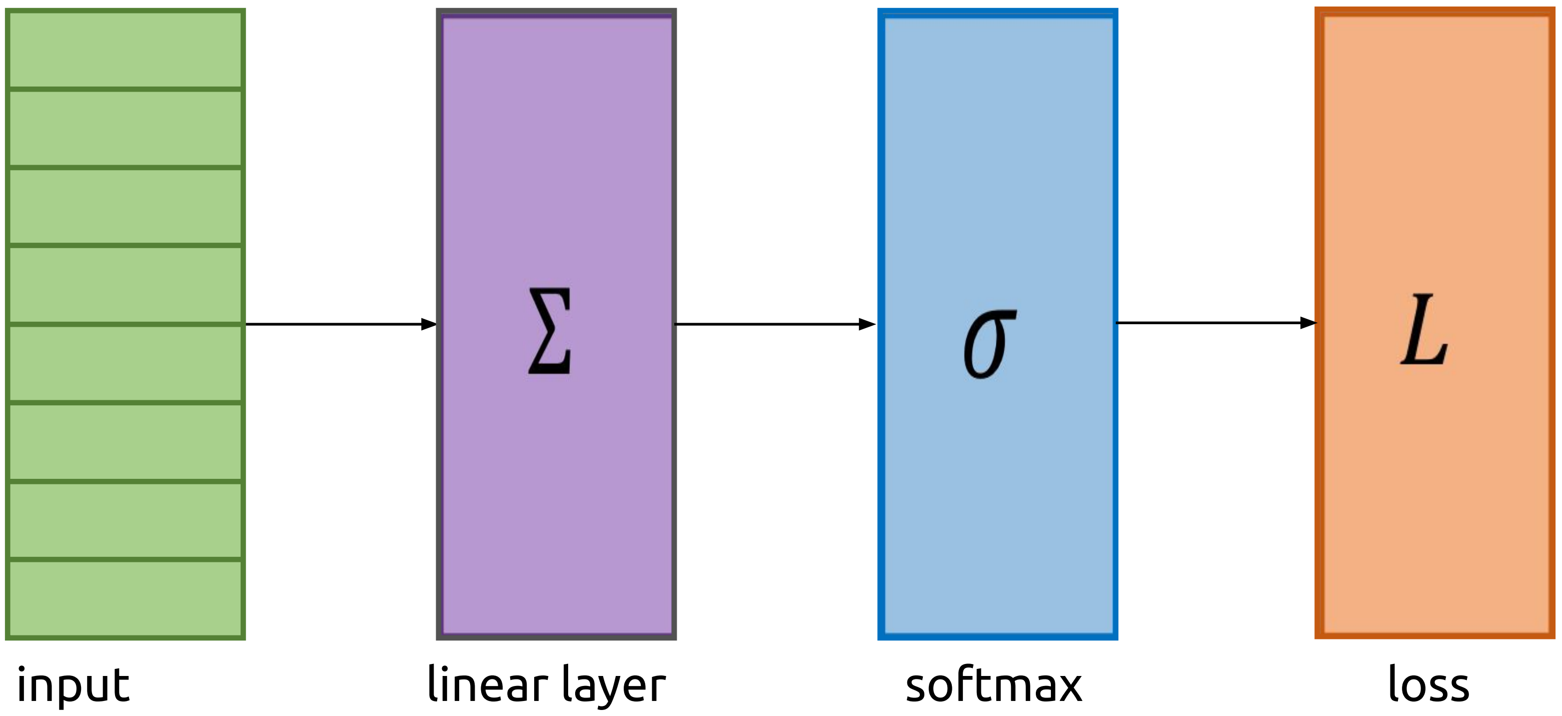


Large batch size:
Slower, stable updates



- Rule of thumb nowadays: Pick the largest batch size you can fit on your GPU!

Generalizing Backpropagation

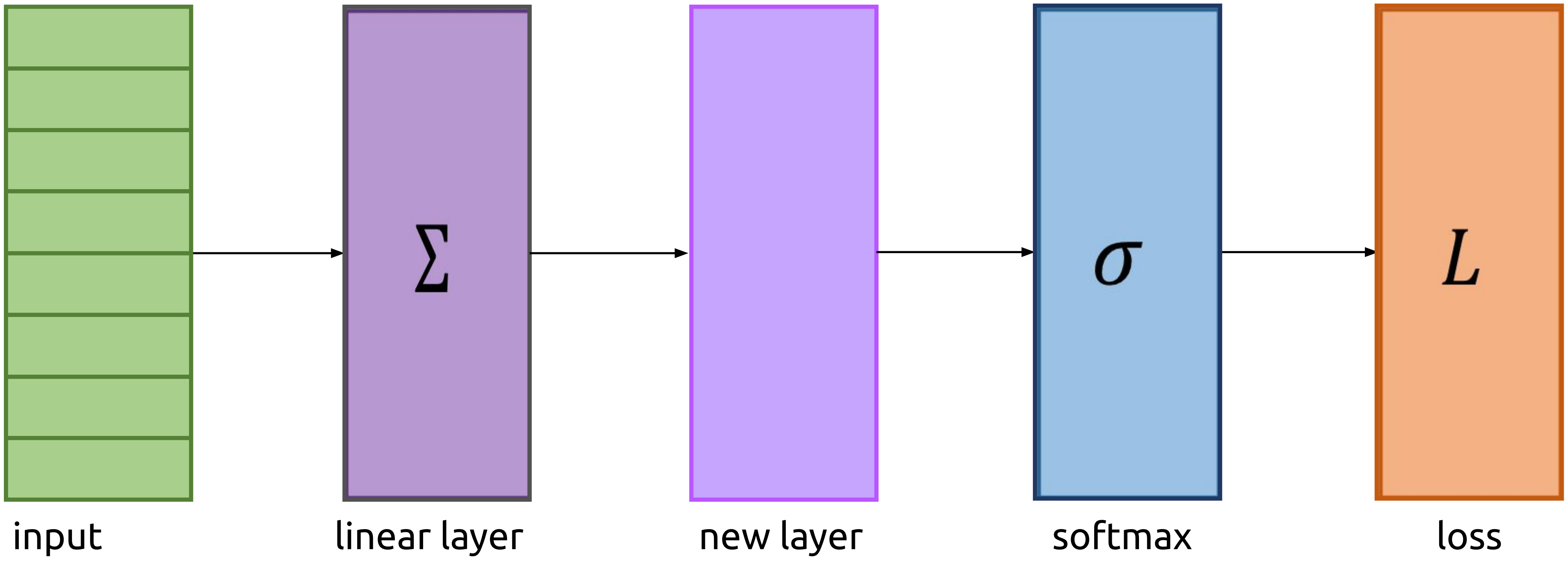


Generalizing Backpropagation

- What if we want to add another layer to our model?
- Calculating derivatives by hand ***again*** is a lot of work ☹️

Can the computers do this for us?

Yes ☺️



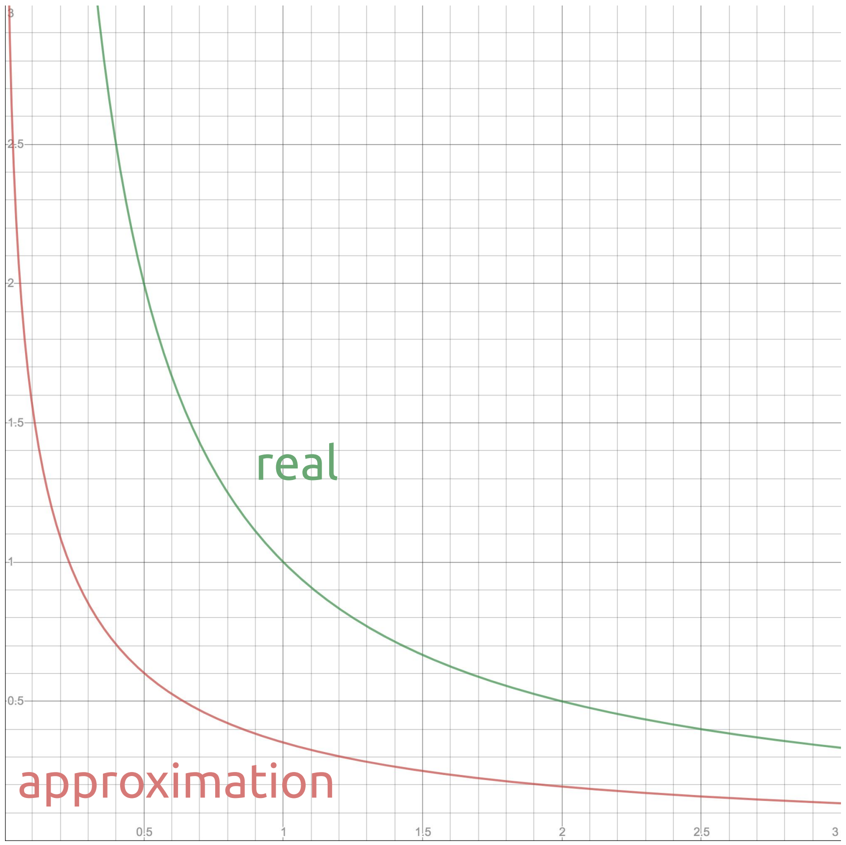
Computer-based Derivatives

- **Numeric differentiation**
  - $\frac{df}{dx} \approx \frac{f(x+\Delta x)-f(x)}{\Delta x}$
  - Pick a small step size Δx
  - Also called "finite differences"

Computer-based Derivatives

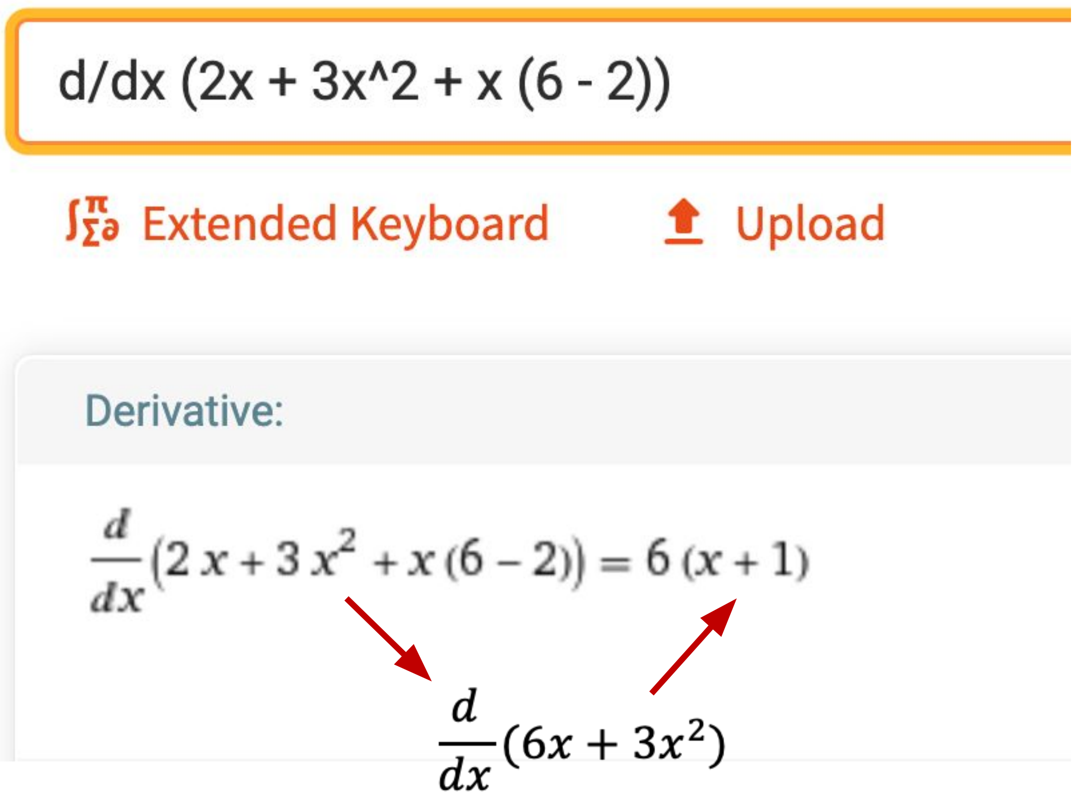
- **Numeric differentiation**
  - $\frac{df}{dx} \approx \frac{f(x+\Delta x)-f(x)}{\Delta x}$
  - Pick a small step size Δx
  - Also called "finite differences"
  - Easy to implement
  - Arbitrarily inaccurate/unstable

$y = \frac{1}{x}$ $\Delta x = 0.5$



Computer-based Derivatives

- Numeric differentiation
- **Symbolic differentiation**
  - Computer "does algebra" and simplifies expressions
  - What Wolfram Alpha does https://www.wolframalpha.com/



Computer-based Derivatives

- Numeric differentiation
- **Symbolic differentiation**
  - Computer "does algebra" and simplifies expressions
  - What Wolfram Alpha does
  - Exact (no approximation error)
  - Complex to implement
  - Only handles static expressions (what about e.g. loops?)

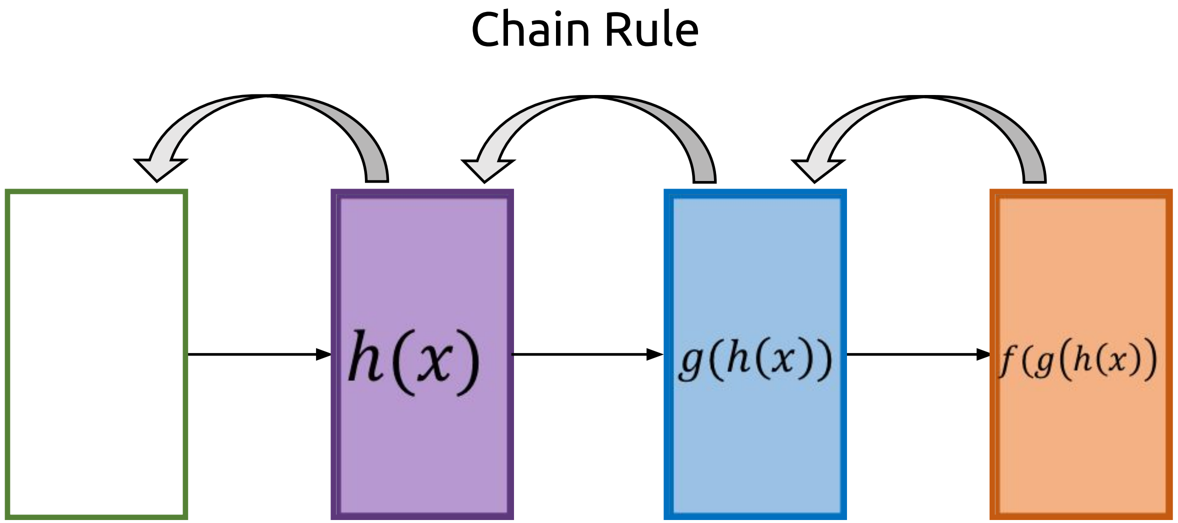
- Example:

```
while abs(x) > 5:
    x = x / 2
```

- This loop could run once or 100 times, it's impossible to know

Computer-based Derivatives

- Numeric differentiation
- Symbolic differentiation
- **Automatic differentiation**
  - Use the chain rule at runtime



Computer-based Derivatives

- Numeric differentiation
- Symbolic differentiation
- **Automatic differentiation**
  - Use the chain rule at runtime
  - Gives exact results
  - Handles dynamics (loops, etc.)
  - Easier to implement
  - Can't simplify expressions

- $\sin^2 x + \cos^2 x \Rightarrow 1$
- Automatic differentiation doesn't know this identity, will end up evaluating the entire expression on the left hand side

Computer-based Derivatives

- Numeric differentiation
- Symbolic differentiation
- **Automatic differentiation**
  - Use the chain rule at runtime
  - Gives exact results
  - Handles dynamics (loops, etc.)
  - Easier to implement
  - Can't simplify expressions
  - What Tensorflow and PyTorch use

- $\sin^2 x + \cos^2 x \Rightarrow 1$
- Automatic differentiation doesn't know this identity, will end up evaluating the entire expression on the left hand side

Two Main "Flavors" of Autodiff

- **Forward Mode Autodiff**
  - Compute derivatives alongside the program as it is running

- **Reverse Mode Autodiff**
  - Run the program, then compute derivatives (in reverse order)

Two Main “Flavors” of Autodiff

- **Forward Mode Autodiff**
  - Compute derivatives alongside the program as it is running

- **Reverse Mode Autodiff**
  - Run the program, then compute derivatives (in reverse order)

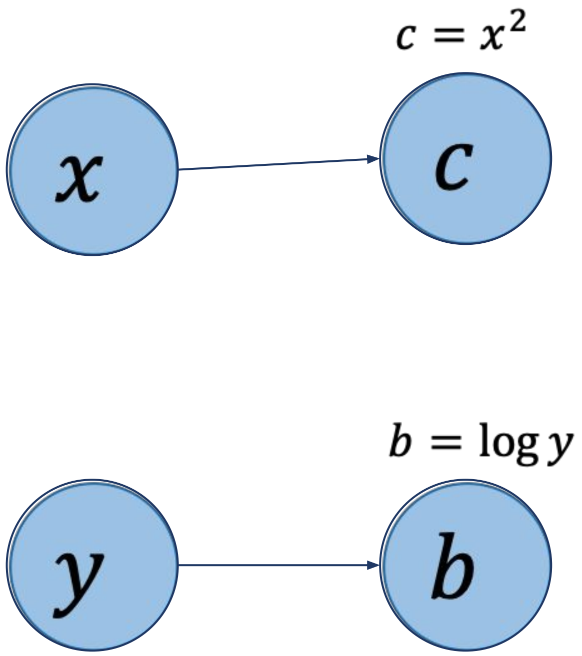
Forward Mode Autodiff

- Given $f(x, y) = x^2 + \log y$

Function inputs

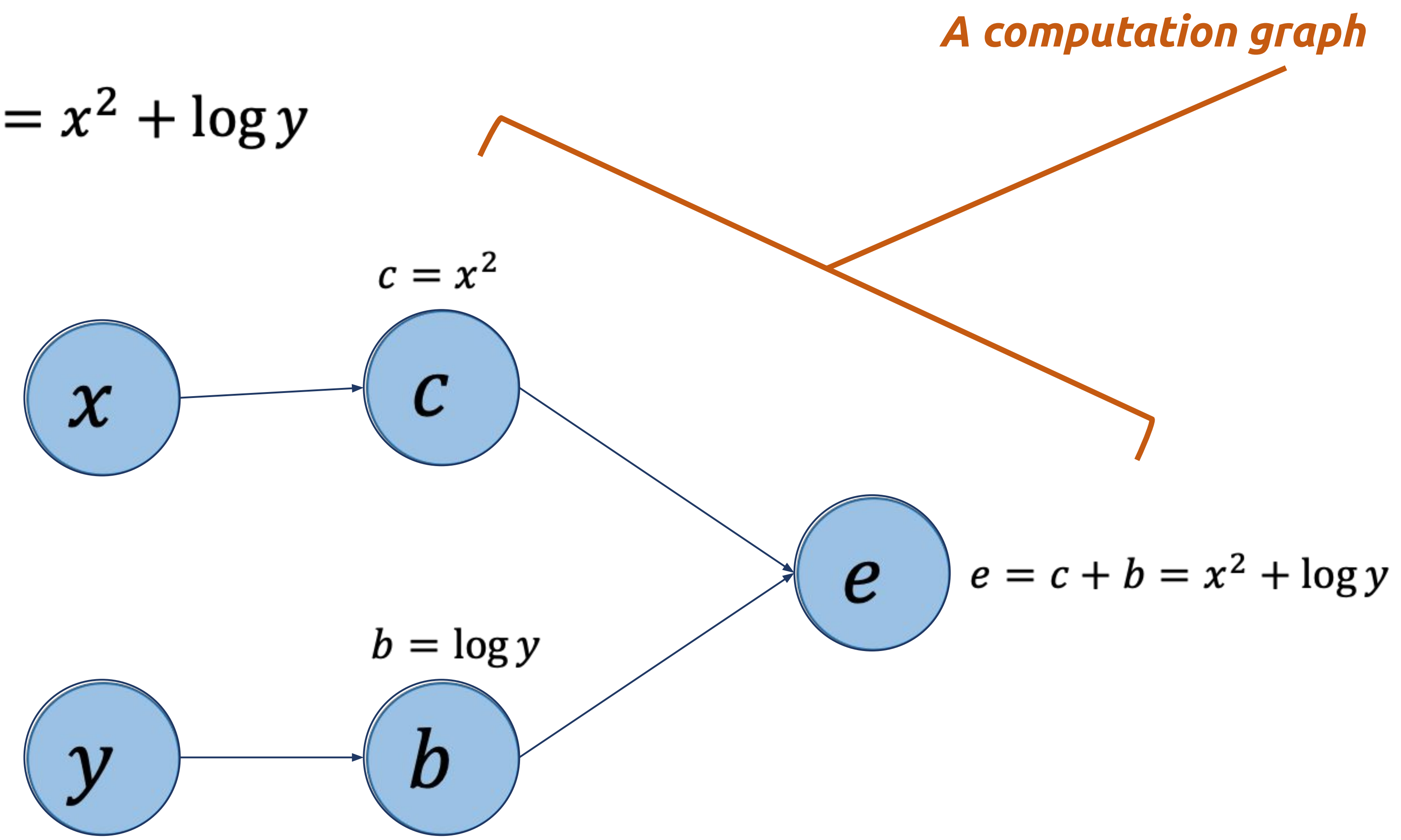
Forward Mode Autodiff

- Given $f(x, y) = x^2 + \log y$

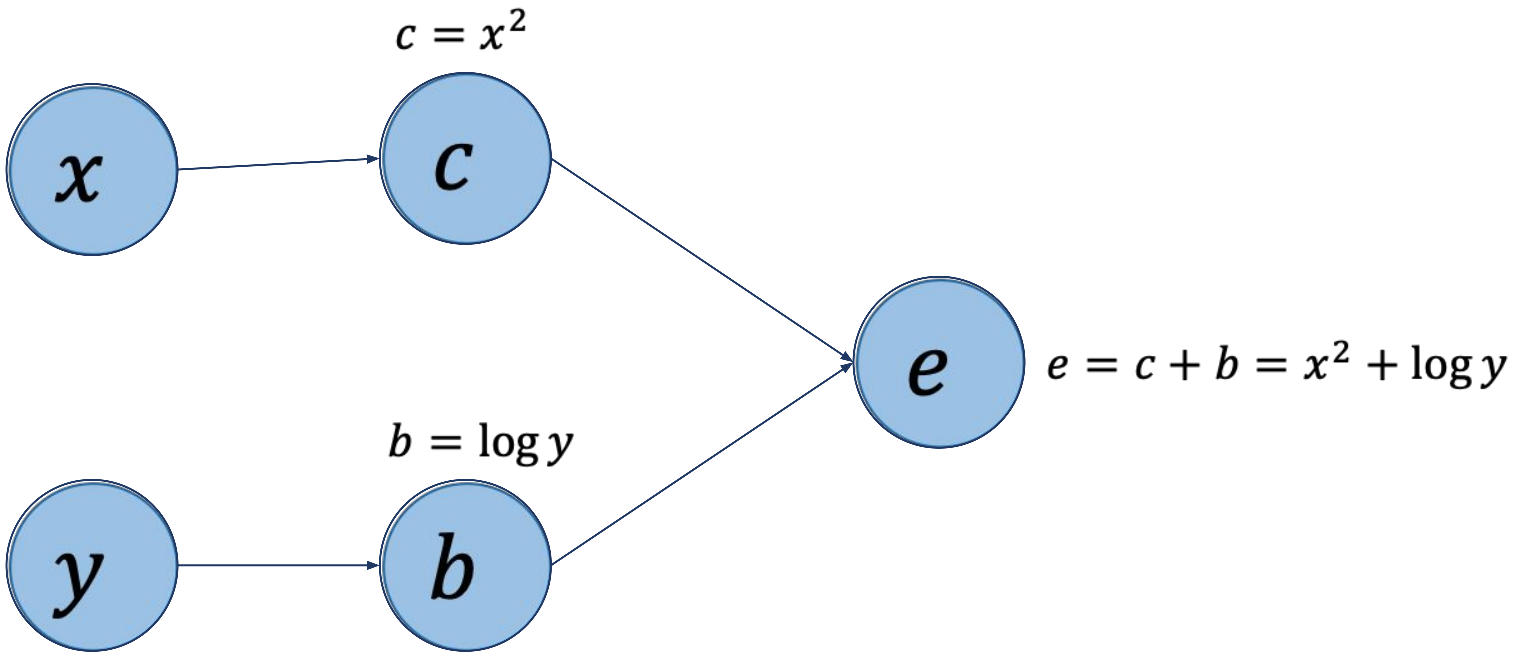


Forward Mode Autodiff

- Given $f(x, y) = x^2 + \log y$

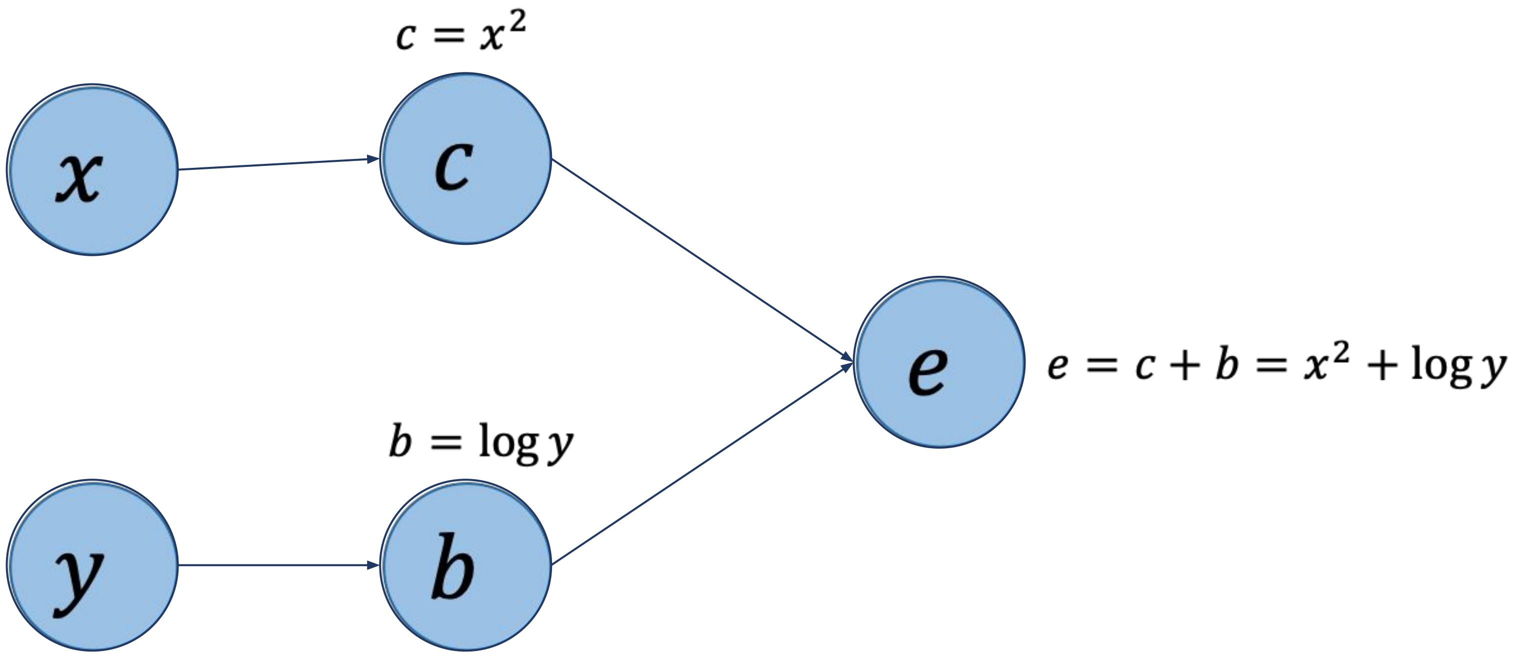


What is the chain rule for $\frac{de}{dx}$ and $\frac{de}{dy}$?



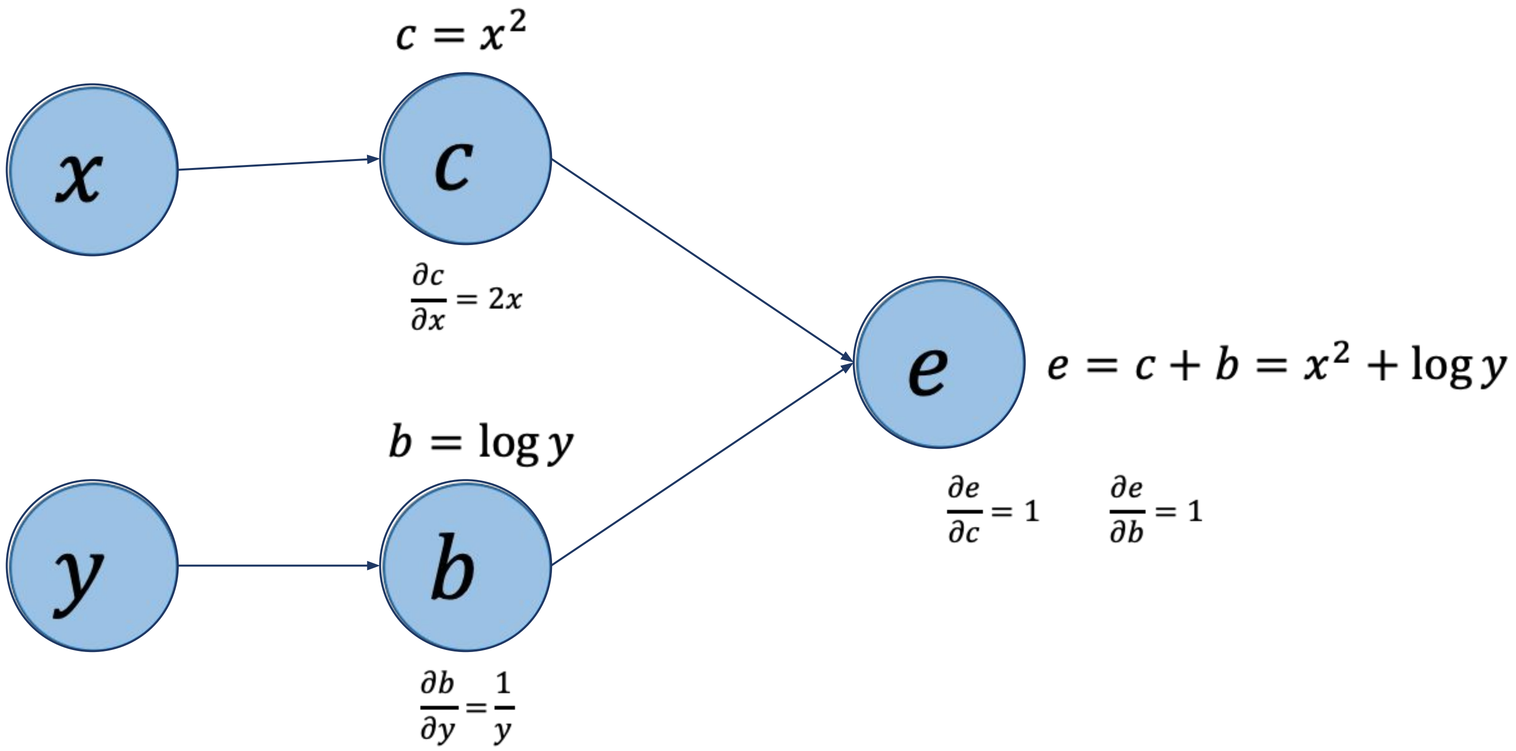
Forward Mode Autodiff

- Idea: Augment each node...



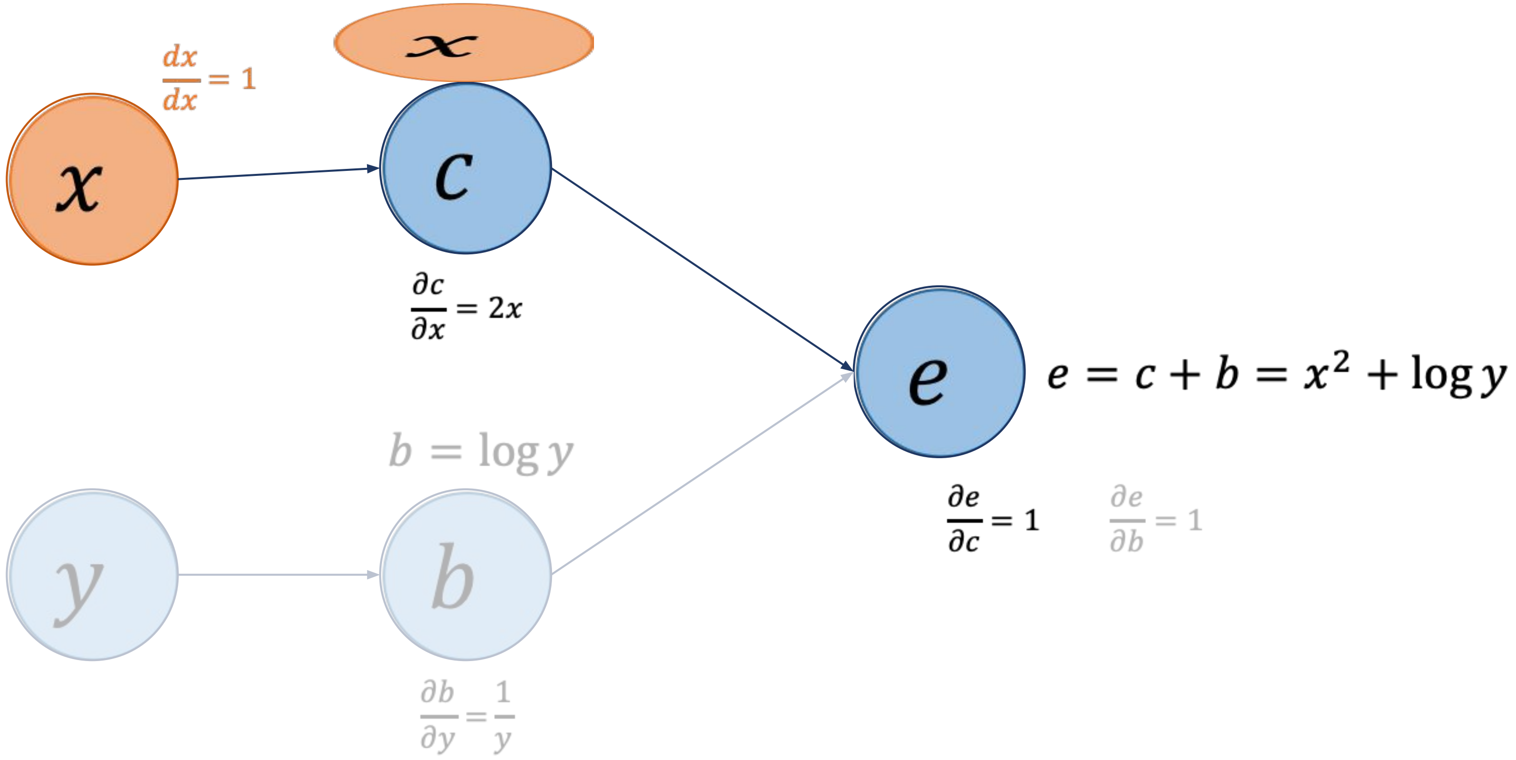
Forward Mode Autodiff

- ...with functions that compute derivatives



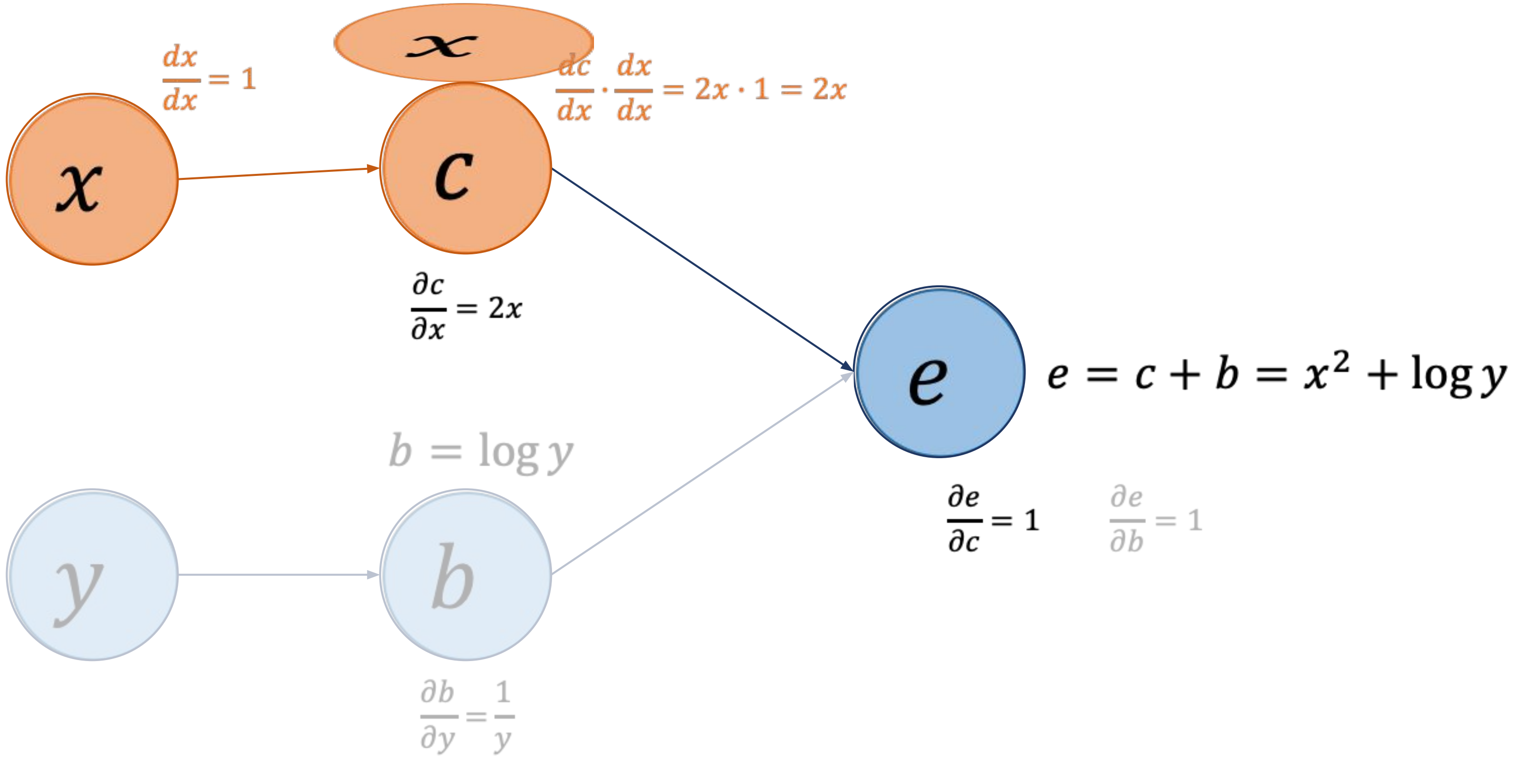
Forward Mode Autodiff

- Then, keep track of derivatives as you compute:



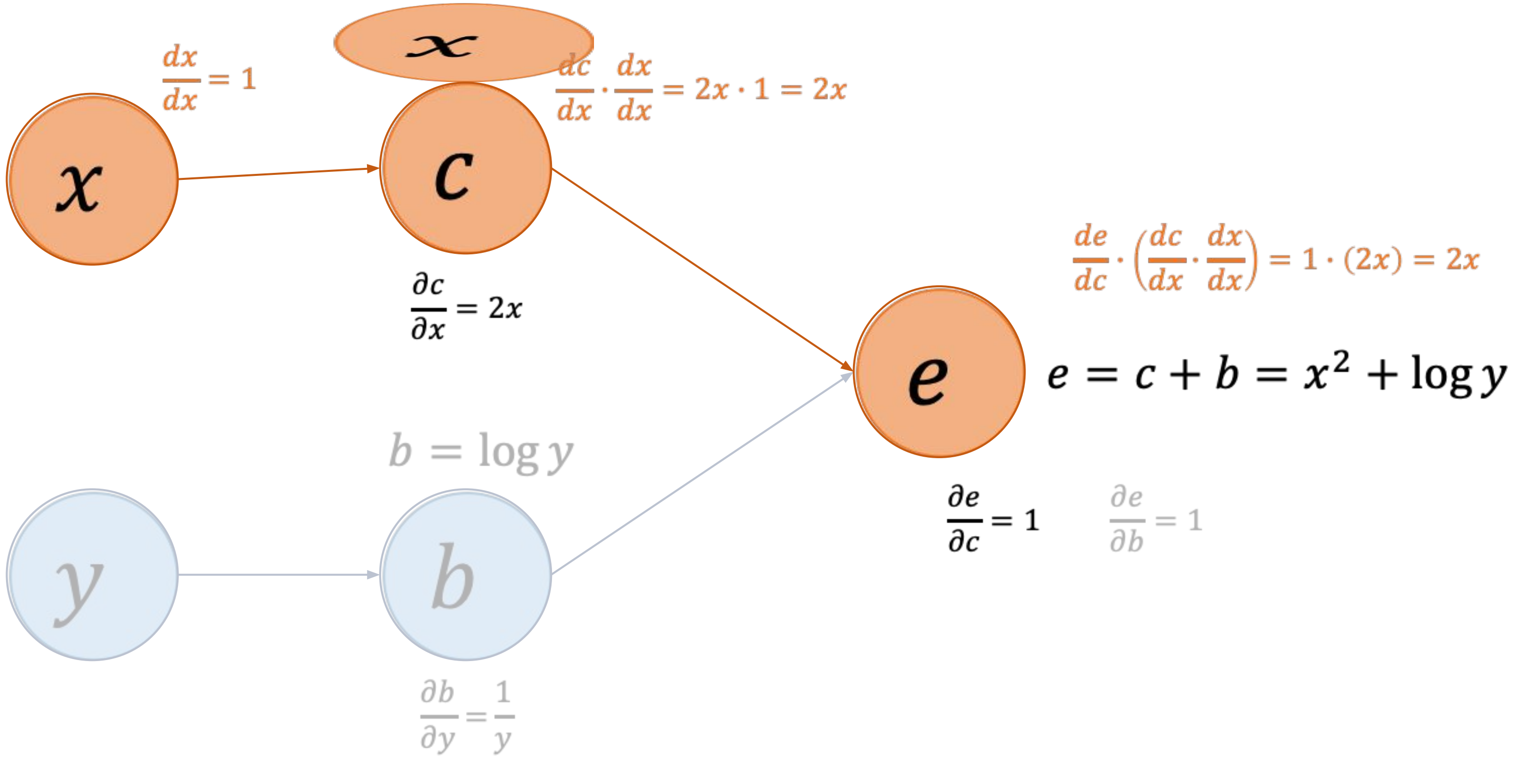
Forward Mode Autodiff

- Then, keep track of derivatives as you compute:



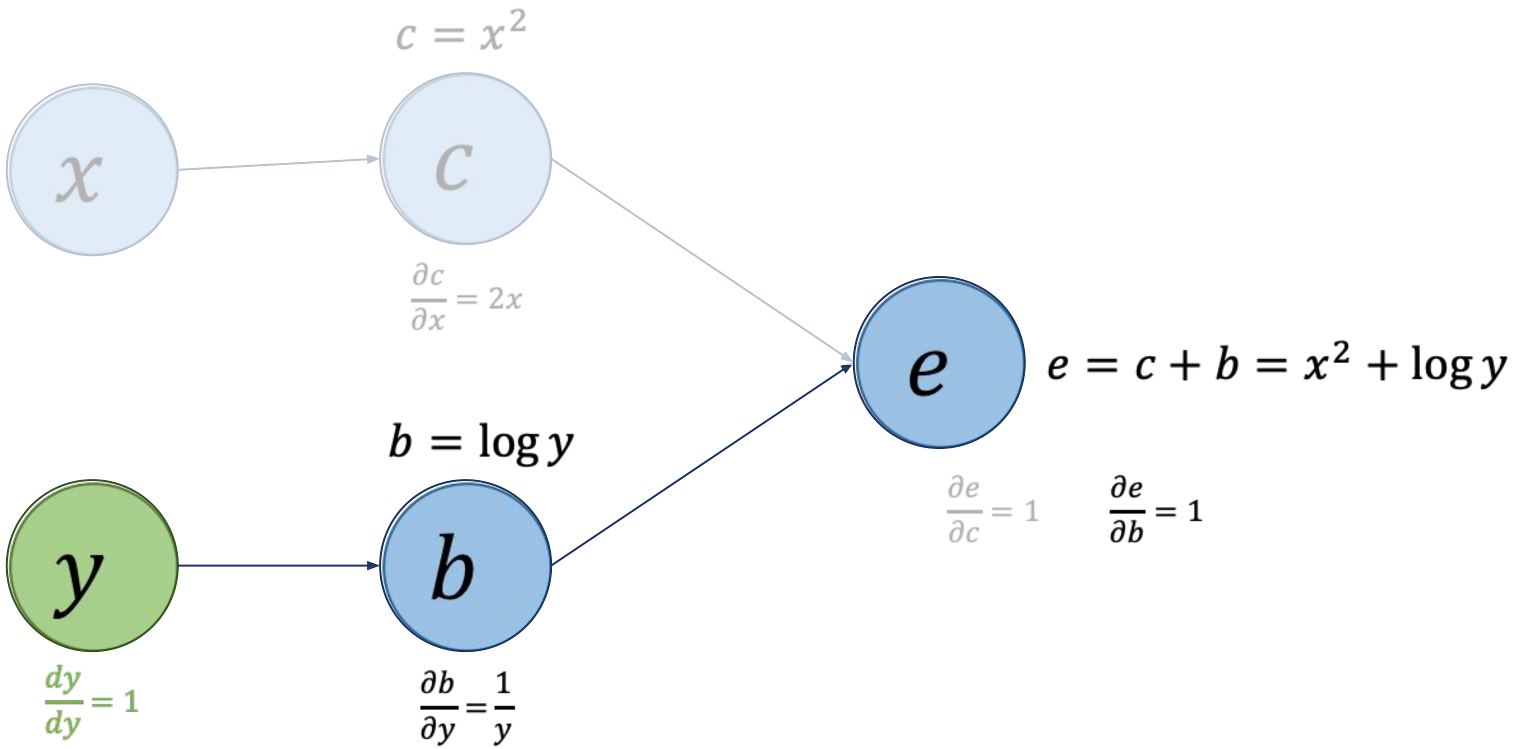
Forward Mode Autodiff

- Then, keep track of derivatives as you compute:



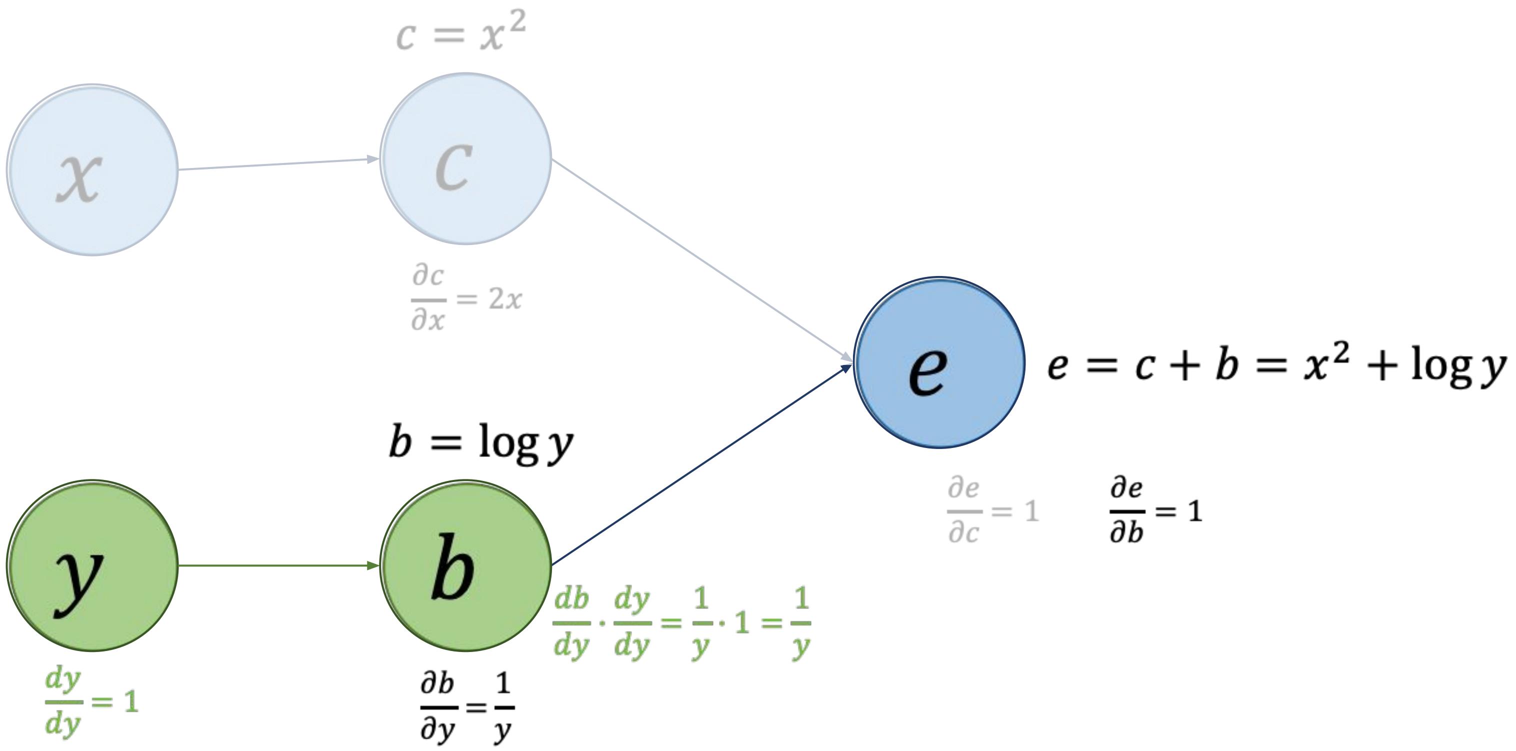
Forward Mode Autodiff

- Can do the same thing starting from the second input:



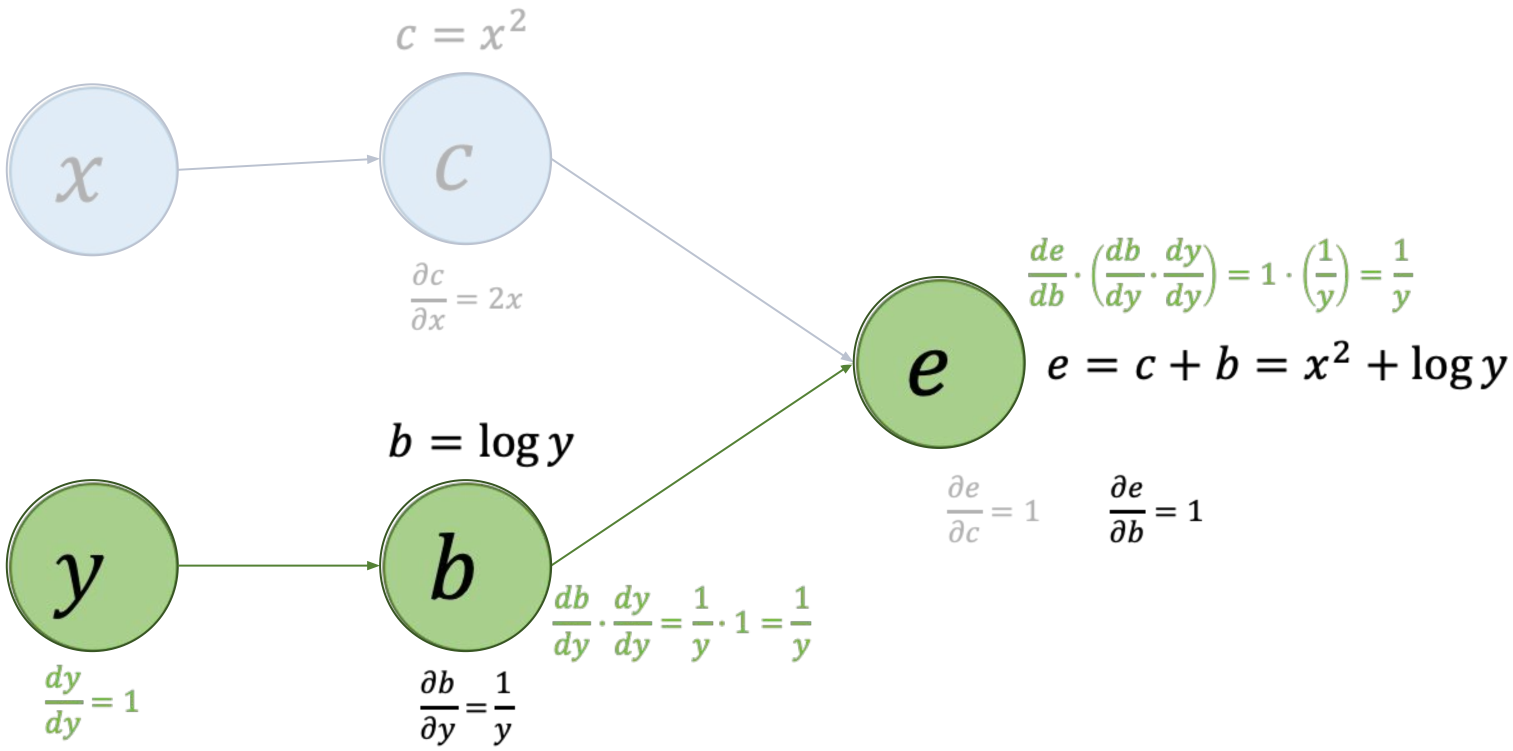
Forward Mode Autodiff

- Can do the same thing starting from the second input:



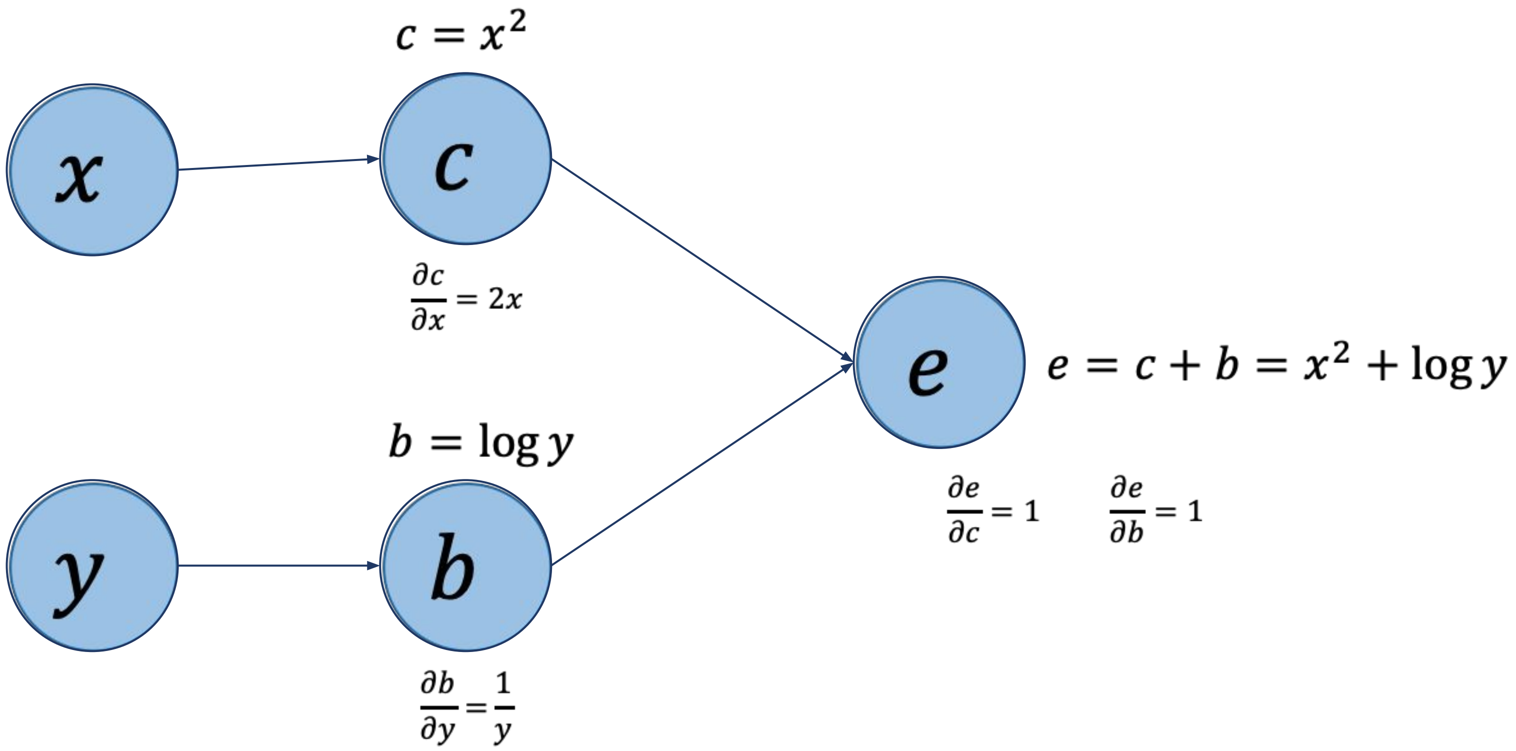
Forward Mode Autodiff

- Can do the same thing starting from the second input:



Forward Mode Autodiff

- We can think of each node...



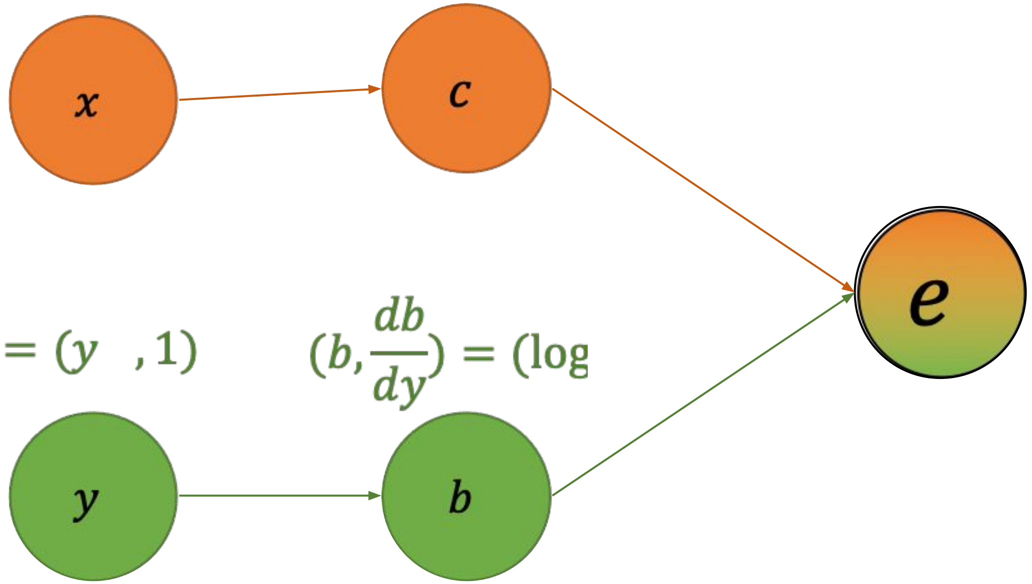
Any questions?

Forward Mode Autodiff

- ...as operating on a (value, derivative) tuple:

These tuples are called **dual numbers**

$(x, \frac{dx}{dx}) = (x \quad , 1)$ $\qquad$ $(c, \frac{dc}{dx}) = (x^2, 2x)$



$(y, \frac{dy}{dy}) = (y \quad , 1)$ $\qquad$ $(b, \frac{db}{dy}) = (\log$

$(e, \frac{de}{dc}) = (c + b, 1)$

$(e, \frac{de}{db}) = (c + b, 1)$

Problems w/ Forward Mode for our use case

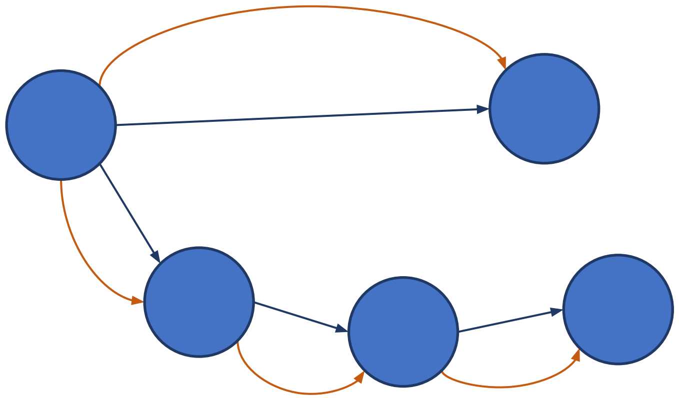
- For $f\colon \mathbb{R} \to \mathbb{R}^n$ (1 input to n outputs) we can differentiate in one pass
- For $f\colon \mathbb{R}^n \to \mathbb{R}$ (n inputs to 1 output) we need n passes

Can you calculate the time and memory complexity?

N = number of input features to the network, K = number of nodes in the graph

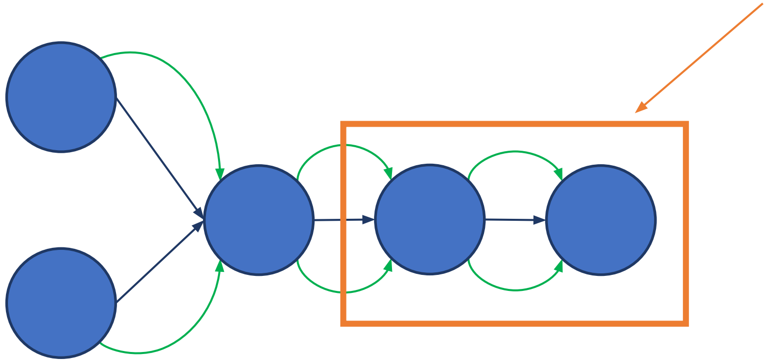
Go to **www.menti.com** and use the code **6413 7504**

$f\colon \mathbb{R} \to \mathbb{R}^n$



$f\colon \mathbb{R}^n \to \mathbb{R}$

these derivatives are being calculated multiple times



Problems w/ Forward Mode for our use case

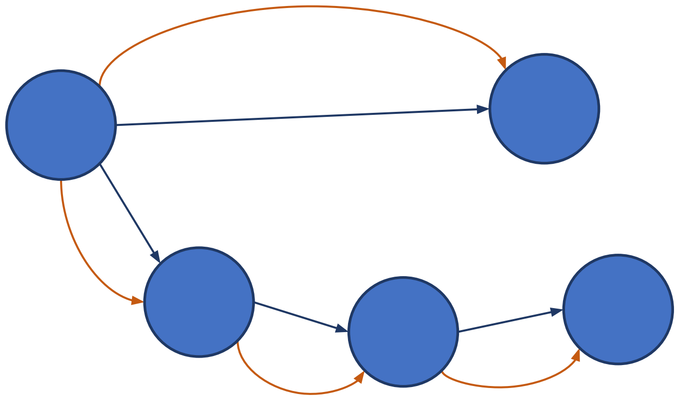
- For $f\colon \mathbb{R} \to \mathbb{R}^n$ (1 input to n outputs) we can differentiate in one pass
- For $f\colon \mathbb{R}^n \to \mathbb{R}$ (n inputs to 1 output) we need n passes

Can you calculate the time and memory complexity?

N = number of input features to the network, K = number of nodes in the graph

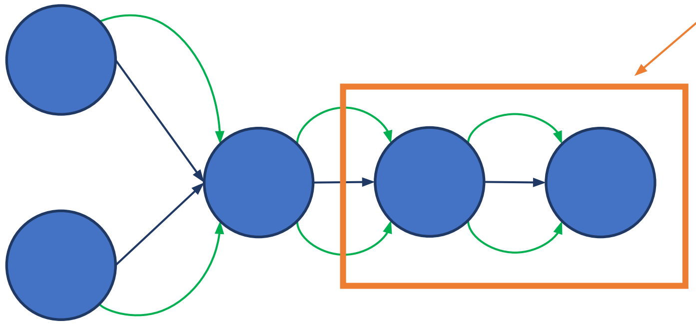
Forward mode: $O(N * K)$ time, $O(1)$ memory

$f\colon \mathbb{R} \to \mathbb{R}^n$



$f\colon \mathbb{R}^n \to \mathbb{R}$

these derivatives are being calculated multiple times



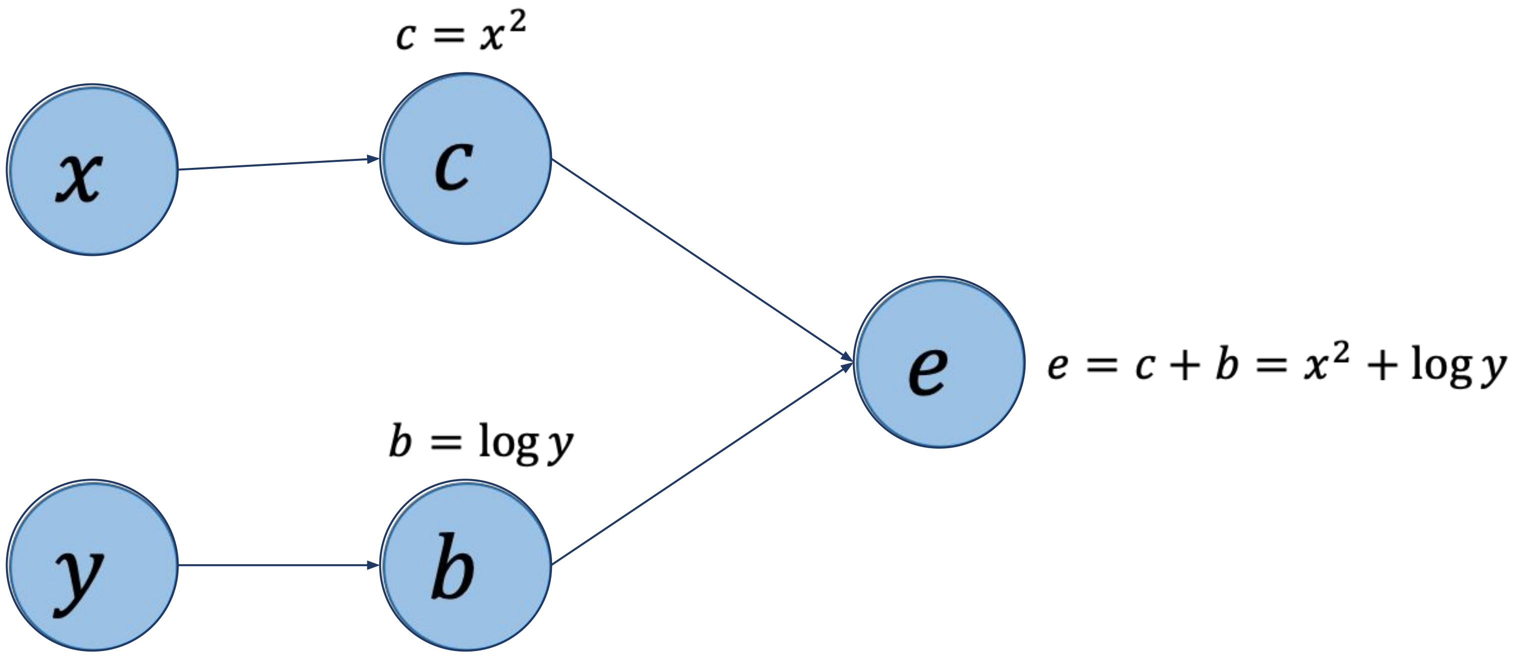
Two Main “Flavors” of Autodiff

- **Forward Mode Autodiff**
  - Compute derivatives alongside the program as it is running

- **Reverse Mode Autodiff**
  - Run the program, then compute derivatives (in reverse order)

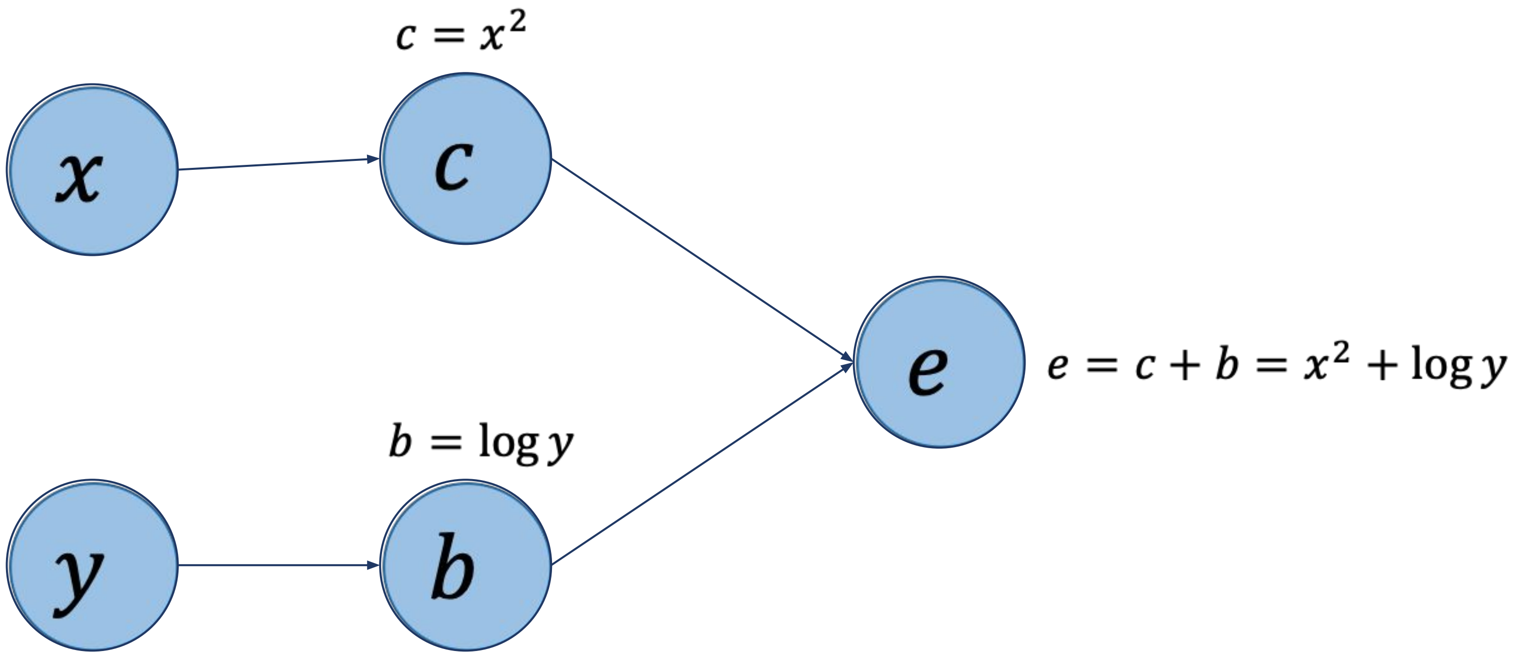
Reverse Mode Autodiff

- Idea: first, run the function forward to produce the graph
- $f(x, y) = x^2 + \log y$



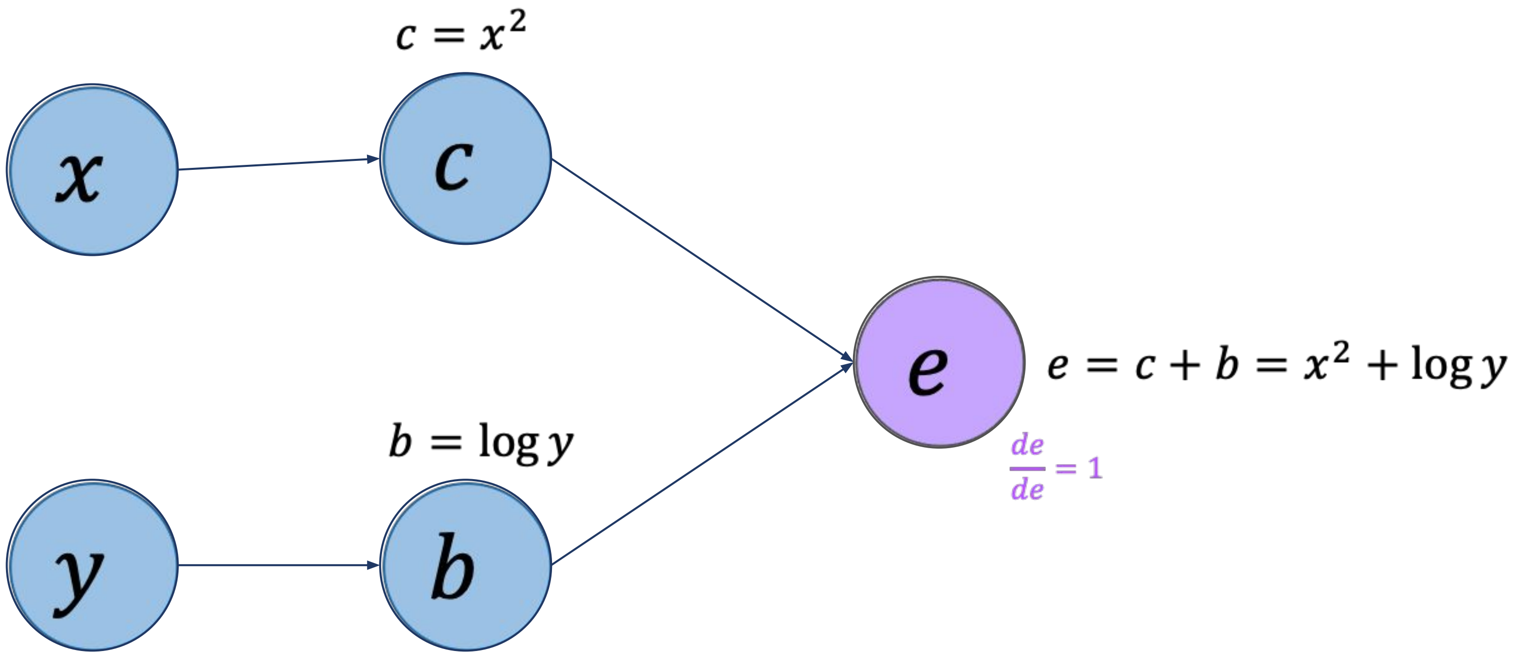
Reverse Mode Autodiff

- Then, compute derivatives ***backward*** from the final node toward the inputs



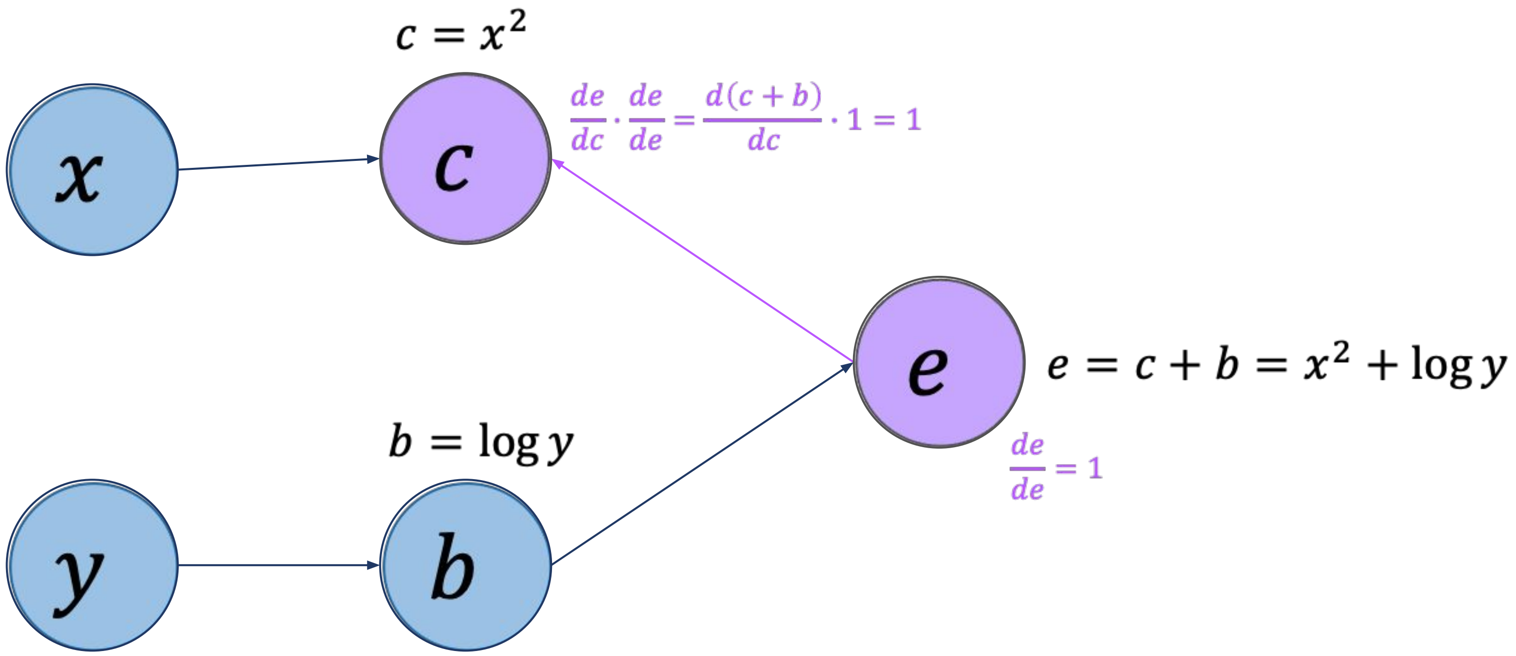
Reverse Mode Autodiff

- Then, compute derivatives ***backward*** from the final node toward the inputs



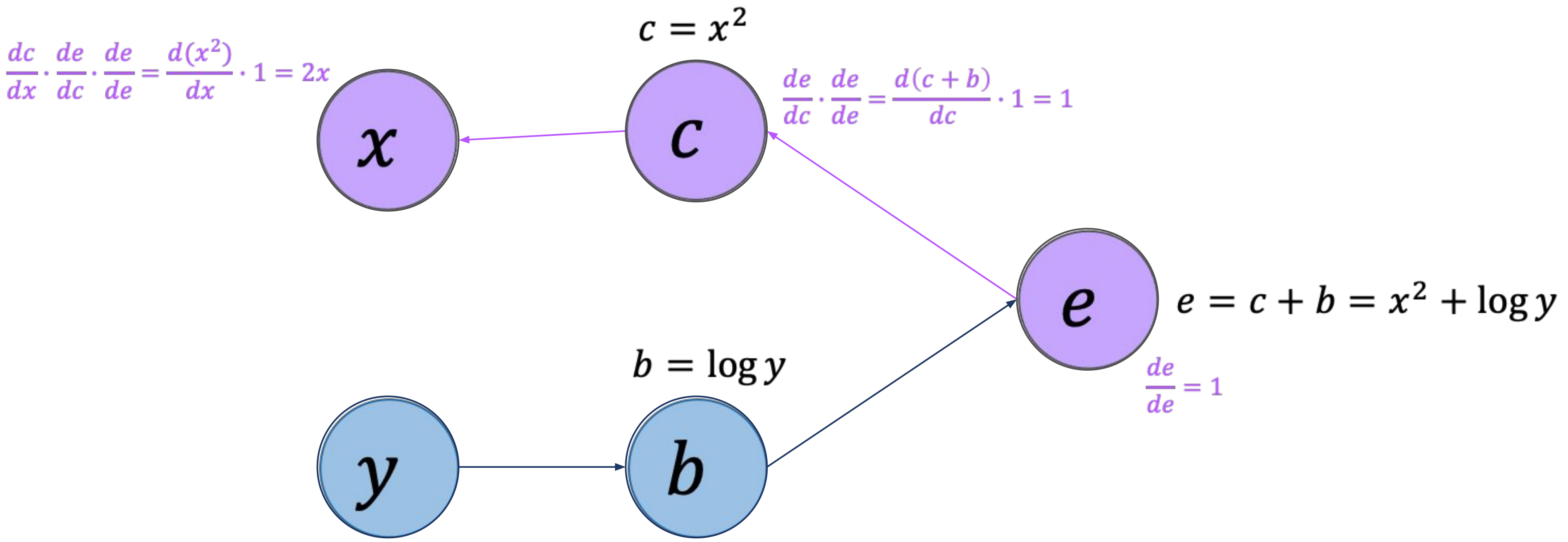
Reverse Mode Autodiff

- Then, compute derivatives ***backward*** from the final node toward the inputs



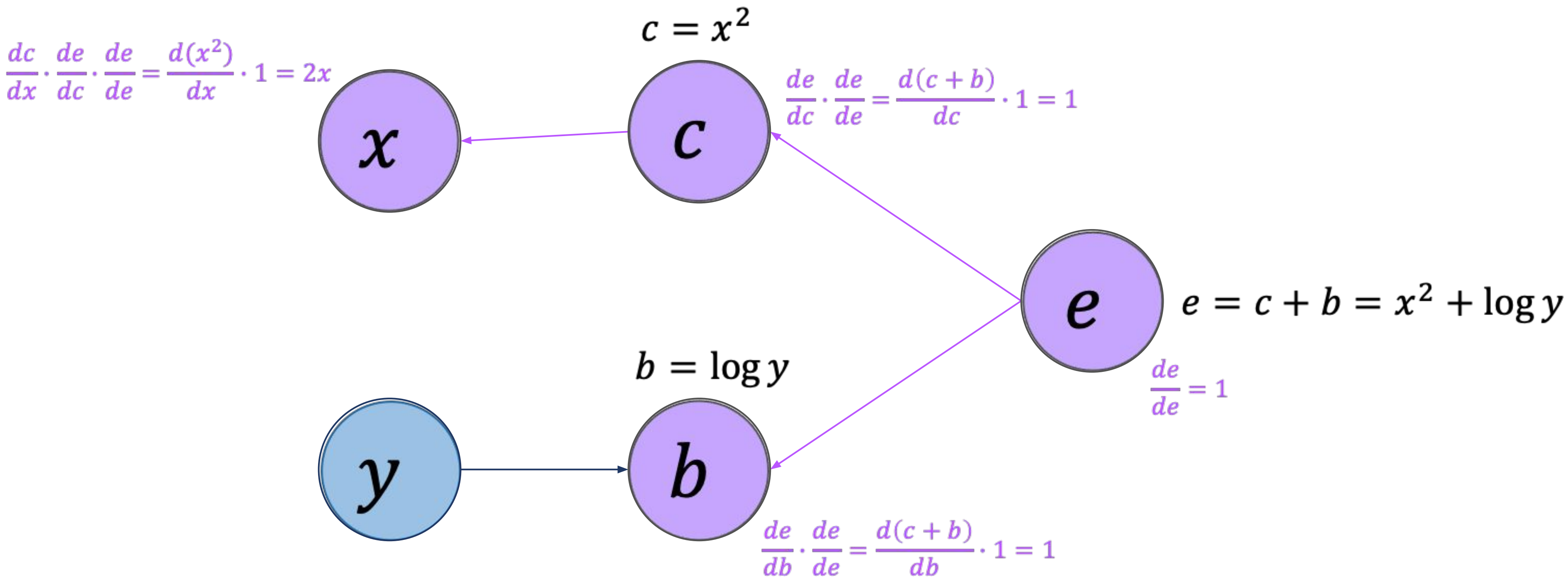
Reverse Mode Autodiff

- Then, compute derivatives ***backward*** from the final node toward the inputs



Reverse Mode Autodiff

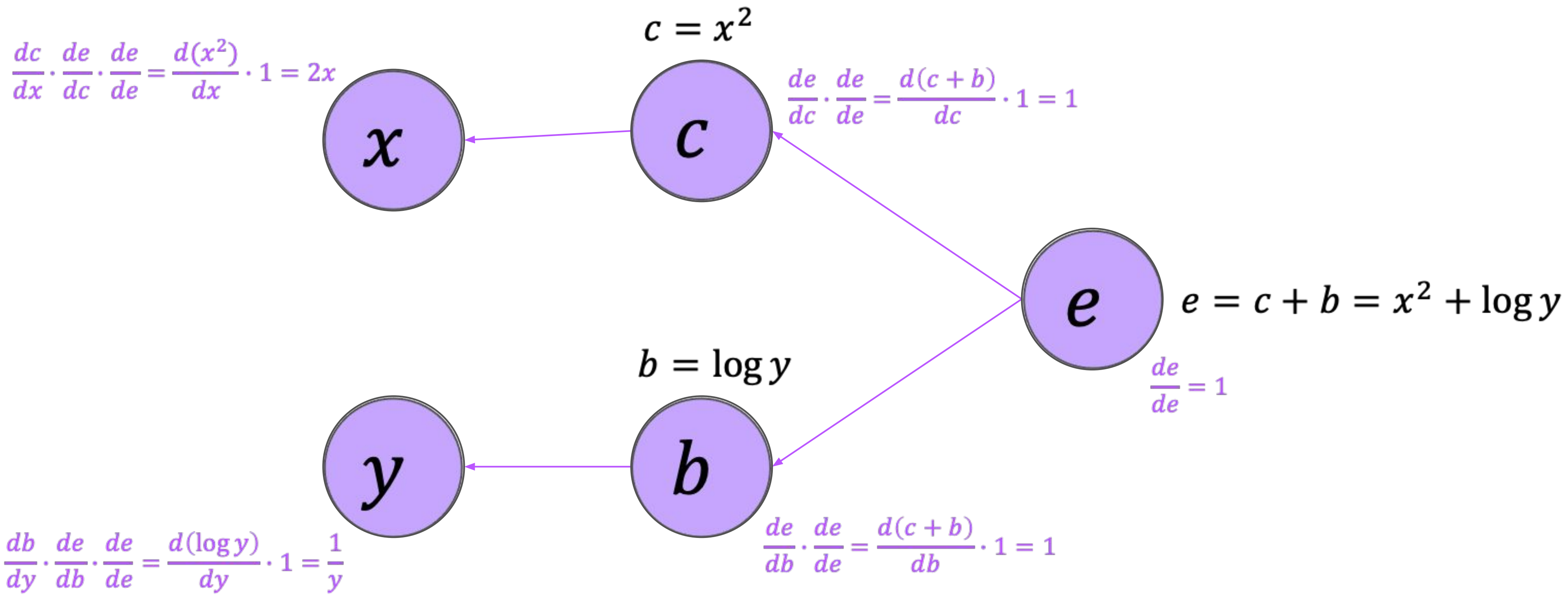
- Then, compute derivatives ***backward*** from the final node toward the inputs



Reverse Mode Autodiff

Can you calculate the time and memory complexity?

- Then, compute derivatives ***backward*** from the final node toward the inputs

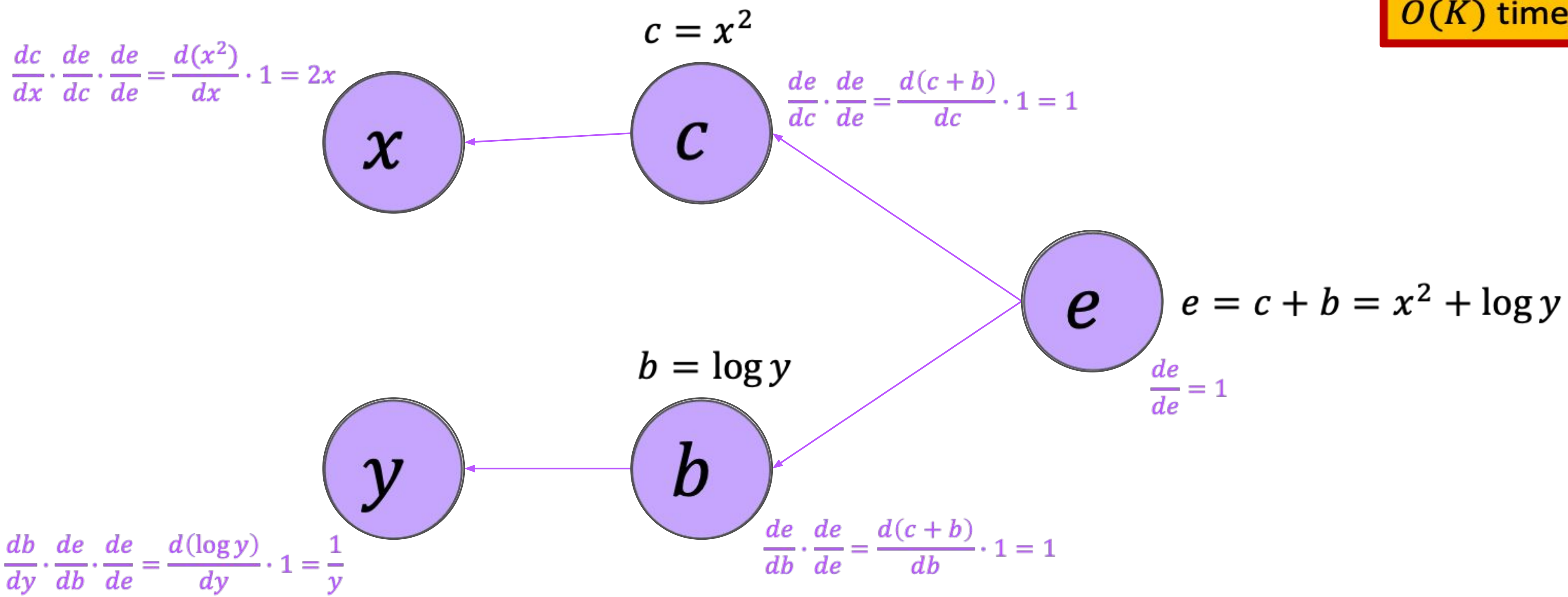


N = number of input features to the network, K = number of nodes in the graph

Reverse Mode Autodiff

- Then, compute derivatives ***backward*** from the final node toward the inputs

$O(K)$ time, $O(K)$ memory



Any questions?

Reverse Mode Autodiff is Time Efficient

- Forward mode: $O(N * K)$ time, $O(1)$ memory
  - $N =$ number of inputs features to the network,
  - $K =$ number of nodes in the graph

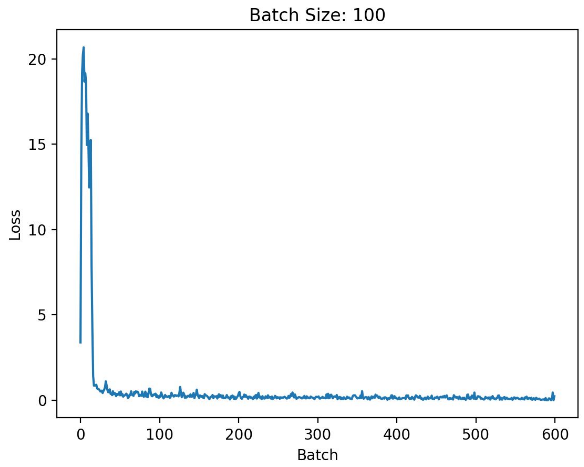
- Reverse mode: $O(K)$ time, $O(K)$ memory

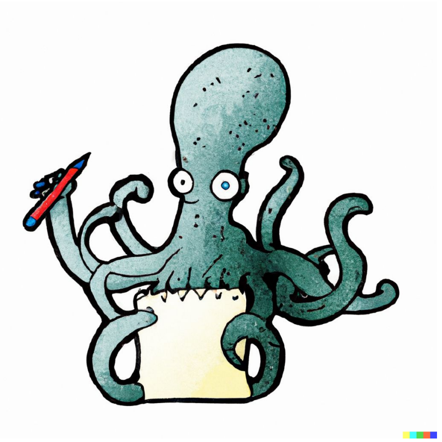
- The memory cost comes from having to keep the entire graph from the forward pass in order to then differentiate backwards

Recap

Computer based derivatives

- Gradient Descent pseudocode
- Stochastic Gradient Descent
- Batching





Deep Learning Frameworks

- Numeric differentiation
- Symbolic differentiation
- Automatic differentiation (Autodiff)
  (1) Forward mode
  (2) Reverse mode