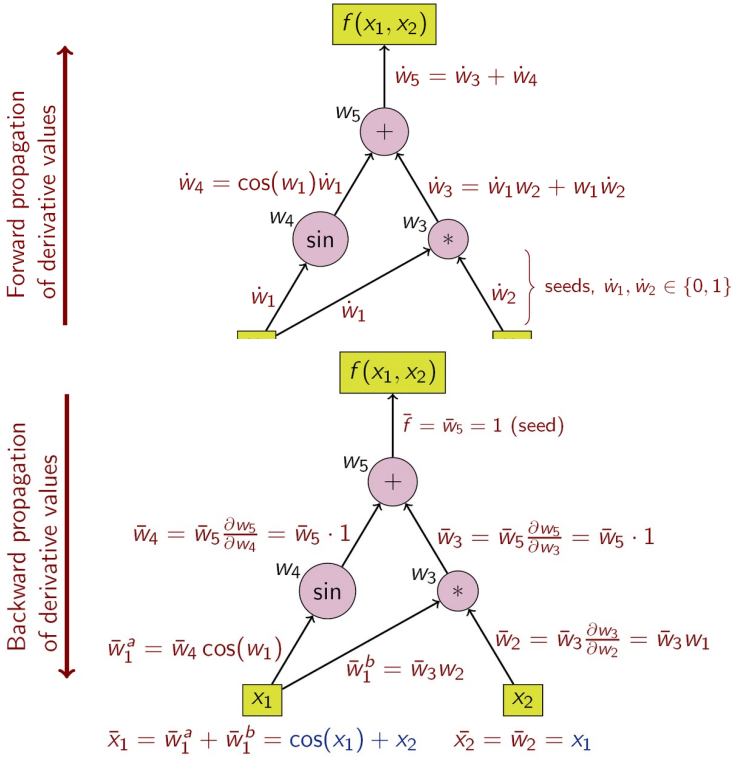
Recitation 3. AutoDiff & Backprop

Sarthak Bisht, Ruimeng Chang

Carnegie Mellon University

Automatic Differentiation Framework

- A set of techniques to evaluate the derivative of a function by repeatedly applying chain rule
- Decompose arbitrarily complex function into simple operations
- Easy to program compared with symbolic differentiation



Credit: Wikipedia

Carnegie Mellon University

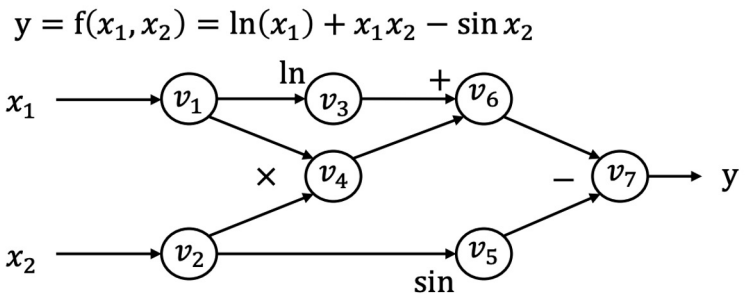
Autodiff Example

$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - \sin(x_2)$

Goal: Given x_1, x_2 as inputs, calculate $\partial y / \partial x_1$

Autodiff Example

$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - \sin(x_2)$



Add intermediate variables

$v_1 = x_1$

$v_2 = x_2$

$v_3 = \ln(v_1)$

$v_4 = v_1 v_2$

$v_5 = \sin(v_2)$

$v_6 = v_3 + v_4$

$y = v_7 = v_6 - v_5$

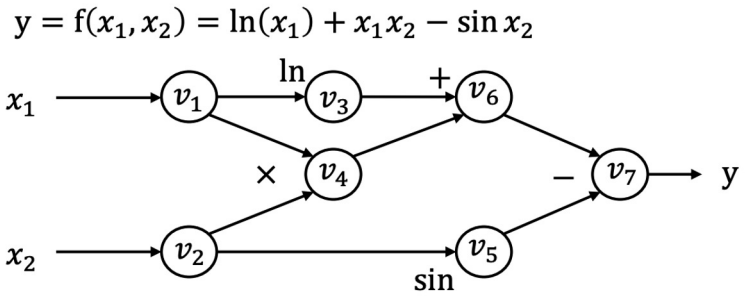
Goal: Given x_1, x_2 as inputs, calculate $\partial y / \partial x_1$

Carnegie Mellon University

Slide from https://dlsyscourse.org/slides/4-automatic-differentiation.pdf

Autodiff Example (Forward)

$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - \sin(x_2)$



Add intermediate variables

$v_1 = x_1$

$v_2 = x_2$

$v_3 = \ln(v_1)$

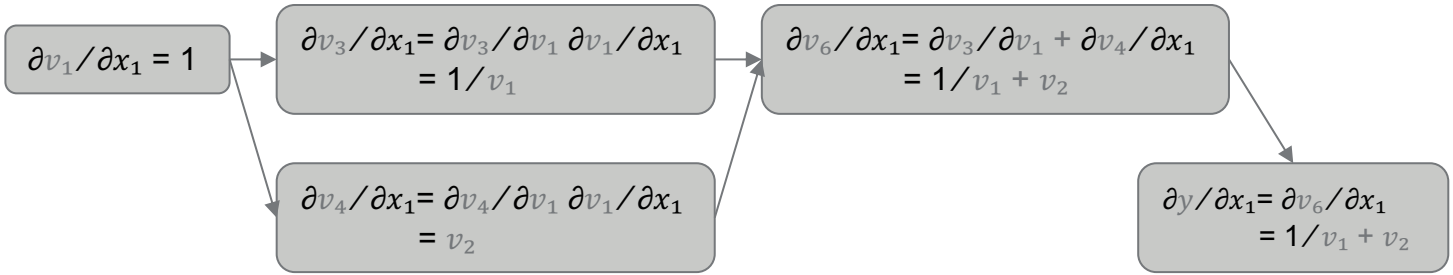
$v_4 = v_1 v_2$

$v_5 = \sin(v_2)$

$v_6 = v_3 + v_4$

$y = v_7 = v_6 - v_5$

Goal: Given x_1, x_2 as inputs, calculate $\partial y / \partial x_1$

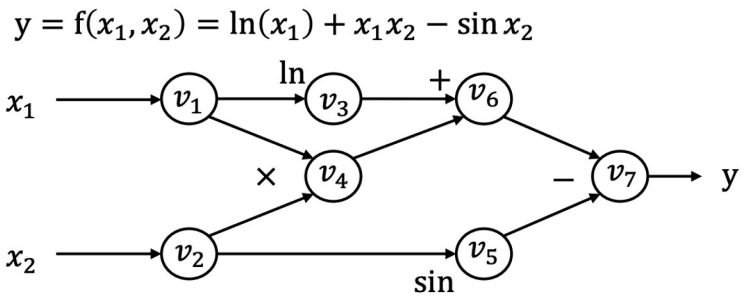


Carnegie Mellon University

Slide from https://dlsyscourse.org/slides/4-automatic-differentiation.pdf

Autodiff Example (Reversed)

$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - \sin(x_2)$



Add intermediate variables

$v_1 = x_1$

$v_2 = x_2$

$v_3 = \ln(v_1)$

$v_4 = v_1 v_2$

$v_5 = \sin(v_2)$

$v_6 = v_3 + v_4$

$y = v_7 = v_6 - v_5$

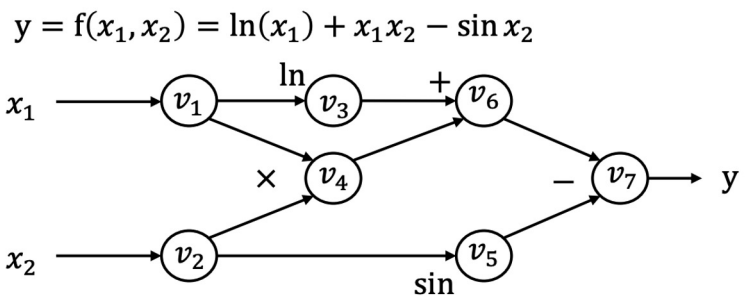
Goal: Given x_1, x_2 as inputs, calculate $\partial y / \partial x_1$

Carnegie Mellon University

Slide from https://dlsyscourse.org/slides/4-automatic-differentiation.pdf

Autodiff Example (Reversed)

$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - \sin(x_2)$



Add intermediate variables

$v_1 = x_1$

$v_2 = x_2$

$v_3 = \ln(v_1)$

$v_4 = v_1 v_2$

$v_5 = \sin(v_2)$

$v_6 = v_3 + v_4$

$y = v_7 = v_6 - v_5$

$\partial y / \partial v_1 = \partial y / \partial v_3 \, \partial v_3 / \partial v_1 + \partial y / \partial v_3 \, \partial v_3 / \partial v_1$
$= 1 / v_1 + v_2$

$\partial y / \partial v_3 = \partial y / \partial v_6 \, \partial v_6 / \partial v_3$
$= 1$

$\partial y / \partial v_6 = \partial y / \partial v_7 \, \partial v_7 / \partial v_6$
$= 1$

$\partial y / \partial v_4 = \partial y / v_6 \, \partial v_6 / \partial v_4$
$= 1$

$\partial y / \partial v_7 = 1$

Carnegie Mellon University

Reference

https://dlsyscourse.org/slides/4-automatic-differentiation.pdf

https://en.wikipedia.org/wiki/Automatic_differentiation

https://deeplearning.cs.cmu.edu/F22/document/recitation/Recitation3/Computing%20Derivatives%20and%20Autograd%20-%20Fall22.pdf

HW1 Autograd

- Optional bonus HW
- Alternative implementation of MyTorch (HW1P1) based on the reversed version of Autodiff framework

What you'll accomplish

- Implement an engine Autograd that stores every operation in sequence (computation graph!)
- Calculate and update gradients for each operation in computation graph
- Build activations, losses, layers using Autograd
- Run MLP using Autograd

Backpropagation

- A special case of Reverse Autodiff where the function output is scalar (i.e. the loss value)
- Part of neural network training
  1. Forward Propagation with current parameters
  2. Calculate the loss
  3. **Backward Propagation to calculate the gradients of the parameters**
  4. Step to update the parameters with gradients

Backpropagation of Loss

- “The function we want to minimize or maximize is called the objective function or criterion. When we are minimizing it, we may also call it the cost function, loss function, or error function.”
- Loss is the starting point of the backpropagation
- Backpropagation is the method to minimize the loss function by adjusting network’s weights and biases. The level of adjustment is determined by the **gradients of the loss function w.r.t. those parameters**.

(Remember: gradient is the transpose of derivative)

Derivative

The derivative of a function f measures the sensitivity of change of the function value w.r.t. a change in its input value x.

$$\frac{dy}{dx} = \lim_{\Delta x \to 0} \frac{\Delta y}{\Delta x}$$

The derivative of y with respect to x:

$$\Delta y = \nabla_x \boldsymbol{y}\, \Delta x$$

Useful formula to check derivative shape!

Note: For scalar y, the derivative wrt x will have the shape as transpose to x

Computing Derivatives – Scalar Chain Rule

$$L = f(z)$$
$$z = g(x)$$

All terms are scalars

$\frac{\partial L}{\partial z}$ is given

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x} = \frac{\partial L}{\partial z} g'(x)$$

Computing Derivatives – Scalar Multiplication

$$L = f(z)$$
$$z = Wx$$

All terms are scalars

$\frac{\partial L}{\partial z}$ is given

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x} = \frac{\partial L}{\partial z} W$$
$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial W} = x\frac{\partial L}{\partial z}$$

Computing Derivatives – Multivariable Chain Rule

$$L = f(z)$$
$$z = g(x)$$

x is D x 1 vector, z is K x 1 vector

$\nabla_z L$ is given (1 x K) matrix

$$\nabla_x L = \nabla_z L \, \nabla_x Z$$

M x D M x K K x D

Computing Derivatives - Multivariate Matrix Multiplication

$$L = f(z)$$
$$z = Wx$$

x is a D x 1 vector

z is a K x 1 vector

W is a K x D matrix

$\nabla_z L$ is given (1 x K) vector

$$\nabla_x L = \nabla_z L \, \nabla_x Z = (\nabla_z L) W \quad \text{1 x D}$$
$$\nabla_W L = \nabla_z L \, \nabla_W Z = x(\nabla_z L) \quad \text{D x K}$$

Computing Derivatives - Multivariable Generalized Chain Rule

$$L = f(z)$$

$$z = z_1 + z_2 + \cdots + z_n = g_1(x) + g_2(x) + \cdots + g_n(x)$$

x is a D x 1 vector

z is a K x 1 vector

$\nabla_z L$ is given (1 x K) matrix

$$\nabla_x L = \nabla_z L \, \nabla_x Z = \nabla_z L(\nabla_x Z_1 + \nabla_x Z_2 + \cdots + \nabla_x Z_n)$$

Questions?

Backward Pass
Why?

- Output layer (N) :
  - For $i = 1 \ldots D_N$
    - $\frac{\partial Div}{\partial y_i^{(N)}} = \frac{\partial Div(Y,d)}{\partial y_i}$
    - $\frac{\partial Div}{\partial z_i^{(N)}} = \frac{\partial Div}{\partial y_i^{(N)}} f_N'\left(z_i^{(N)}\right)$
    - $\frac{\partial Div}{\partial w_{ij}^{(N)}} = y_i^{(N-1)} \frac{\partial Div}{\partial z_j^{(N)}}$ for $j = 0 \ldots D_{N-1}$
- For layer $k = N-1$ *downto* 1
  - For $i = 1 \ldots D_k$
    - $\frac{\partial Div}{\partial y_i^{(k)}} = \sum_j w_{ij}^{(k+1)} \frac{\partial Di}{\partial z_j^{(k+1)}}$
    - $\frac{\partial Div}{\partial z_i^{(k)}} = \frac{\partial Div}{\partial y_i^{(k)}} f_k'\left(z_i^{(k)}\right)$
    - $\frac{\partial Div}{\partial w_{ij}^{(k)}} = y_i^{(k-1)} \frac{\partial Div}{\partial z_j^{(k)}}$ for $j = 0 \ldots D_{k-1}$

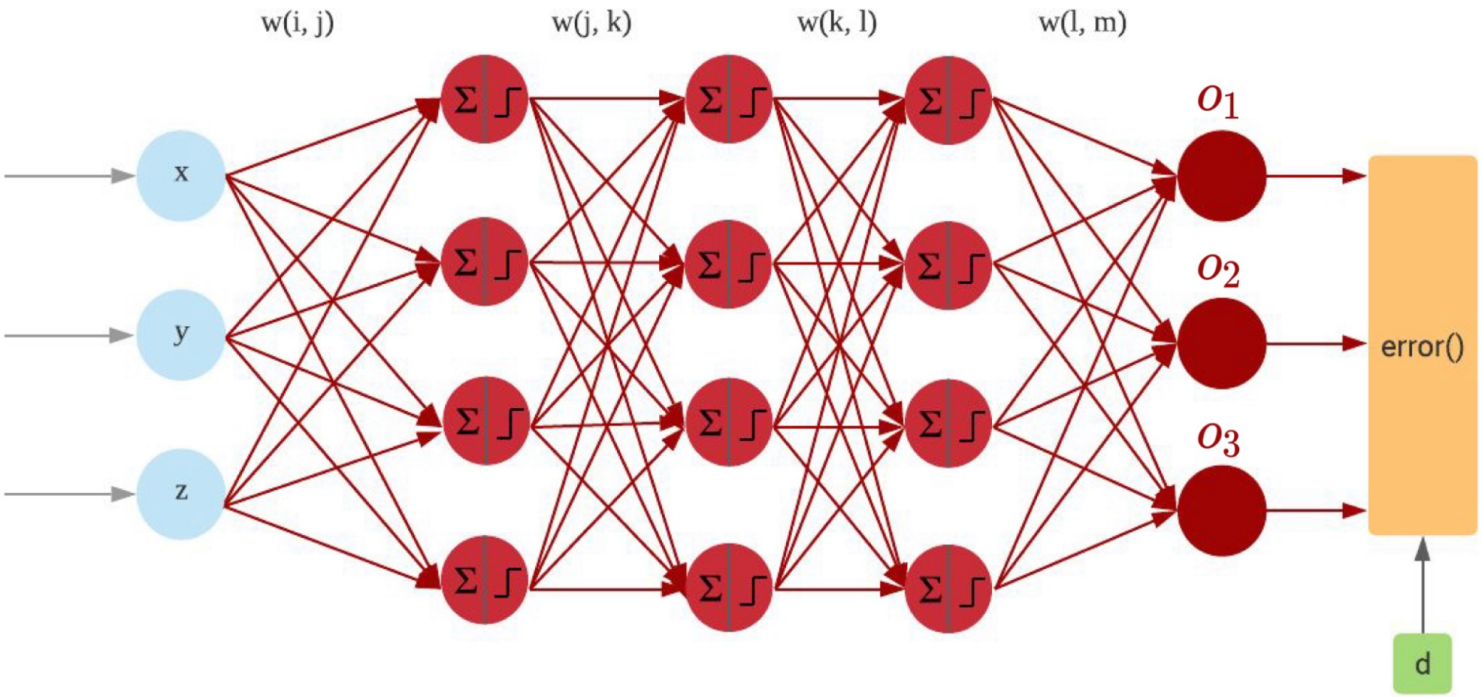
Called "Backpropagation" because the derivative of the loss is propagated "backwards" through the network

Very analogous to the forward pass:

Backward weighted combination of next layer

Backward equivalent of activation

Why calculate partial derivatives in backward direction?



Carnegie Mellon University

AutoDiff vs Numerical Differentiation?

Directly compute the partial gradient by definition

$$\frac{\partial f(\theta)}{\partial \theta_i} = \lim_{\epsilon \to 0} \frac{f(\theta + \epsilon e_i) - f(\theta)}{\epsilon}$$

A more numerically accurate way to approximate the gradient

$$\frac{\partial f(\theta)}{\partial \theta_i} = \frac{f(\theta + \epsilon e_i) - f(\theta - \epsilon e_i)}{2\epsilon} + o(\epsilon^2)$$

Suffer from numerical error, less efficient to compute

Carnegie Mellon University

Slide from https://dlsyscourse.org/slides/4-automatic-differentiation.pdf

AutoDiff vs Symbolic Differentiation?

Write down the formulas, derive the gradient by sum, product and chain rules

$$\frac{\partial(f(\theta)+g(\theta))}{\partial\theta} = \frac{\partial f(\theta)}{\partial\theta} + \frac{\partial g(\theta)}{\partial\theta} \qquad \frac{\partial(f(\theta)g(\theta))}{\partial\theta} = \mathrm{g}(\theta)\frac{\partial f(\theta)}{\partial\theta} + \mathrm{f}(\theta)\frac{\partial g(\theta)}{\partial\theta} \qquad \frac{\partial f(g(\theta))}{\partial\theta} = \frac{\partial f(g(\theta))}{\partial g(\theta)}\frac{\partial g(\theta)}{\partial\theta}$$

Naively do so can result in wasted computations

Example: $f(\theta) = \prod_{i=0}^{n} \theta_i \qquad \frac{f(\theta)}{\partial\theta_k} = \prod_{j\neq k}^{n} \theta_j$

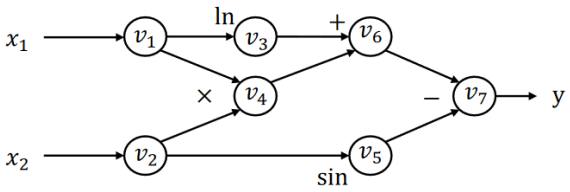
Cost $n(n-1)$ multiplies to compute all partial gradients

Carnegie Mellon University

Slide from https://dlsyscourse.org/slides/4-automatic-differentiation.pdf

What's the difference?

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - \sin x_2$$



Forward Mode AutoDiff

Define $\dot{v}_i = \frac{\partial v_i}{\partial x_1}$

We can then compute the $\dot{v}_i$ iteratively in the forward topological order of the computational graph

Forward AD trace

$$\dot{v}_1 = 1$$
$$\dot{v}_2 = 0$$
$$\dot{v}_3 = \dot{v}_1 / v_1 = 0.5$$
$$\dot{v}_4 = \dot{v}_1 v_2 + \dot{v}_2 v_1 = 1 \times 5 + 0 \times 2 = 5$$
$$\dot{v}_5 = \dot{v}_2 \cos v_2 = 0 \times \cos 5 = 0$$
$$\dot{v}_6 = \dot{v}_3 + \dot{v}_4 = 0.5 + 5 = 5.5$$
$$\dot{v}_7 = \dot{v}_6 - \dot{v}_5 = 5.5 - 0 = 5.5$$

Now we have $\frac{\partial y}{\partial x_1} = \dot{v}_7 = 5.5$

Reverse Mode AutoDiff

Define adjoint $\overline{v_i} = \frac{\partial y}{\partial v_i}$

We can then compute the $\overline{v_i}$ iteratively in the **reverse** topological order of the computational graph

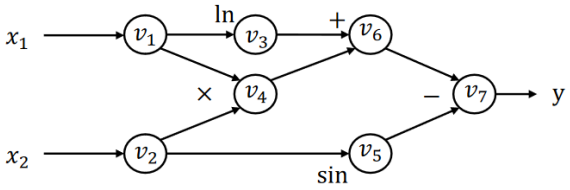
Reverse AD evaluation trace

$$\overline{v_7} = \frac{\partial y}{\partial v_7} = 1$$
$$\overline{v_6} = \overline{v_7}\frac{\partial v_7}{\partial v_6} = \overline{v_7} \times 1 = 1$$
$$\overline{v_5} = \overline{v_7}\frac{\partial v_7}{\partial v_5} = \overline{v_7} \times (-1) = -1$$
$$\overline{v_4} = \overline{v_6}\frac{\partial v_6}{\partial v_4} = \overline{v_6} \times 1 = 1$$
$$\overline{v_3} = \overline{v_6}\frac{\partial v_6}{\partial v_3} = \overline{v_6} \times 1 = 1$$
$$\overline{v_2} = \overline{v_5}\frac{\partial v_5}{\partial v_2} + \overline{v_4}\frac{\partial v_4}{\partial v_2} = \overline{v_5} \times \cos v_2 + \overline{v_4} \times v_1 = -0.284 + 2 = 1.716$$
$$\overline{v_1} = \overline{v_4}\frac{\partial v_4}{\partial v_1} + \overline{v_3}\frac{\partial v_3}{\partial v_1} = \overline{v_4} \times v_2 + \overline{v_3}\frac{1}{v_1} = 5 + \frac{1}{2} = 5.5$$

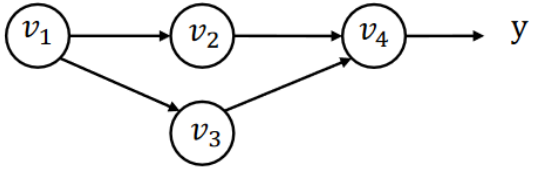
Slide from https://dlsyscourse.org/slides/4-automatic-differentiation.pdf

Multiple Pathway Case

$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - \sin x_2$



v_1 is being used in multiple pathways (v_2 and v_3)



y can be written in the form of $y = f(v_2, v_3)$

$$\overline{v_1} = \frac{\partial y}{\partial v_1} = \frac{\partial f(v_2,v_3)}{\partial v_2}\frac{\partial v_2}{\partial v_1} + \frac{\partial f(v_2,v_3)}{\partial v_3}\frac{\partial v_3}{\partial v_1} = \overline{v_2}\frac{\partial v_2}{\partial v_1} + \overline{v_3}\frac{\partial v_3}{\partial v_1}$$

Define partial adjoint $\overline{v_{i\to j}} = \bar{v}_j \dfrac{\partial v_j}{\partial v_i}$ for each input output node pair i and j

$$\bar{v}_i = \sum_{j \in next(i)} \overline{v_{i\to j}}$$

We can compute partial adjoints separately then sum them together

Reverse AutoDiff Algorithm

```
def gradient(out):
    node_to_grad = {out: [1]}

    for i in reverse_topo_order(out):
        v̄_i = Σ_j v̄_{i→j} = sum(node_to_grad[i])

        for k ∈ inputs(i):
            compute v̄_{k→i} = v̄_i ∂v_i/∂v_k
            append v̄_{k→i} to node_to_grad[k]

    return adjoint of input v̄_input
```

Dictionary that records a list of partial adjoints of each node

Sum up partial adjoints

"Propagates" partial adjoint to its input

Slide from https://dlsyscourse.org/slides/4-automatic-differentiation.pdf

Reverse AutoDiff by Extending Computational Graph

```
def gradient(out):
  node_to_grad = {out:  [1]}
  for i in reverse_topo_order(out):
    v̄_i = Σ_j v̄_{i→j} = sum(node_to_grad[i])
    for k ∈ inputs(i):
        compute v̄_{k→i} = v̄_i ∂v_i/∂v_k
        append v̄_{k→i} to node_to_grad[k]
    return adjoint of input v̄_input
```

$i = 2$

node_to_grad: {

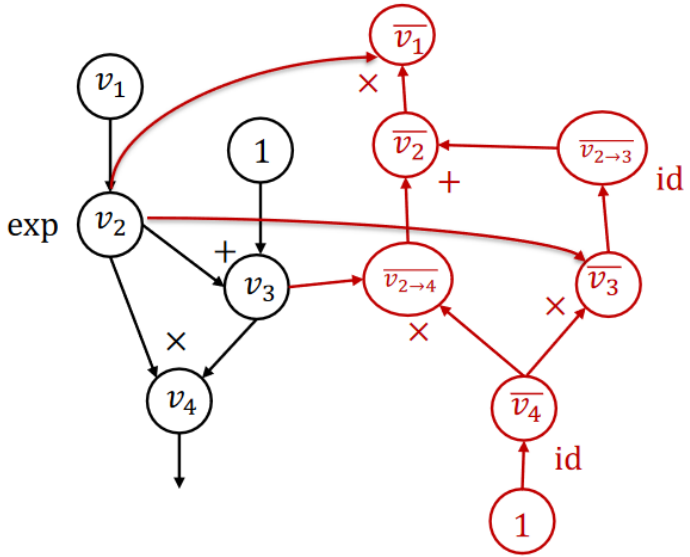
1: $[\overline{v_1}]$

2: $[\overline{v_{2\to4}}, \overline{v_{2\to3}}]$

3: $[\overline{v_3}]$

4: $[\overline{v_4}]$

}



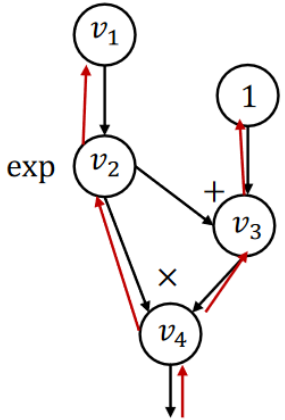
NOTE: id is identity function

Carnegie Mellon University

Slide from https://dlsyscourse.org/slides/4-automatic-differentiation.pdf

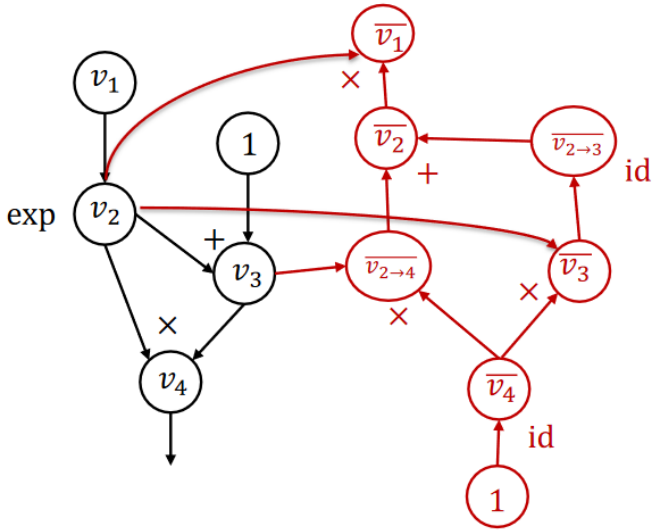
Backprop vs Reverse mode AutoDiff

Backprop



- Run backward operations the same forward graph
- Used in first generation deep learning frameworks (caffe, cuda-convnet)

Reverse mode AD by extending computational graph



- Construct separate graph nodes for adjoints
- Used by modern deep learning frameworks

Carnegie Mellon University

Slide from https://dlsyscourse.org/slides/4-automatic-differentiation.pdf