CS6216 Advanced Topics in Machine Learning (Systems)

Automatic Differentiation

Yao LU

28 Aug 2024

National University of Singapore
School of Computing

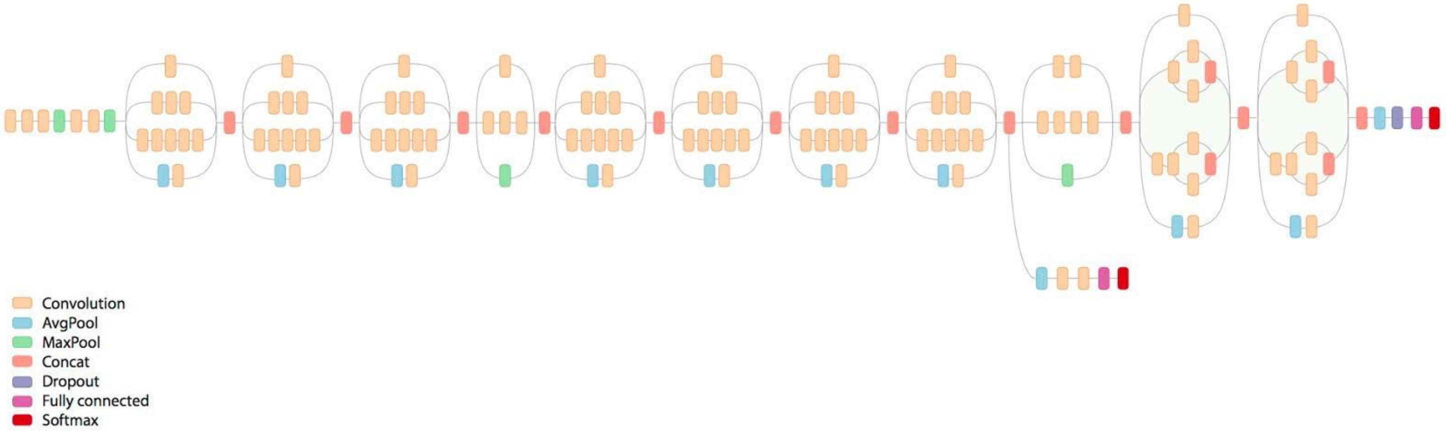
Recap: algorithmic workflows

Stochastic Gradient Descent (SGD)

Train ML models through many iterations of 3 stages

1. **Forward propagation**: apply model to a batch of input samples and run calculation through operators to produce a prediction
2. **Backward propagation**: run the model in reverse to produce partial gradients / errors for each trainable weight
3. **Weight update:** use the loss value to update model weights $w \leftarrow w - \eta \nabla_w L(w)$

Model inputs



Prediction

Loss

Label

Ways to compute gradients

- Numerical differentiation
- Symbolic differentiation
- Forward mode automatic differentiation
- Backward mode automatic differentiation

Numerical Differentiation

- Directly compute the partial gradient by symbolic definitions

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

⇒Hard to work correctly due to precision / numerical errors

Symbolic Differentiation

- Use the model formula to derive gradients by sum, product and chain rules
  - $\frac{\partial\left(f(\theta)+g(\theta)\right)}{\partial(\theta)} =$
  - $\frac{\partial\left(f(\theta)g(\theta)\right)}{\partial(\theta)} =$
  - $\frac{\partial f(g(\theta))}{\partial(\theta)} =$

$\Rightarrow$ Lots of repeated compute: $f(\theta) = \prod_{i=1}^{n} \theta_i$, $\frac{f(\theta)}{\partial\theta_k} = \prod_{j\neq k}^{n} \theta_j$

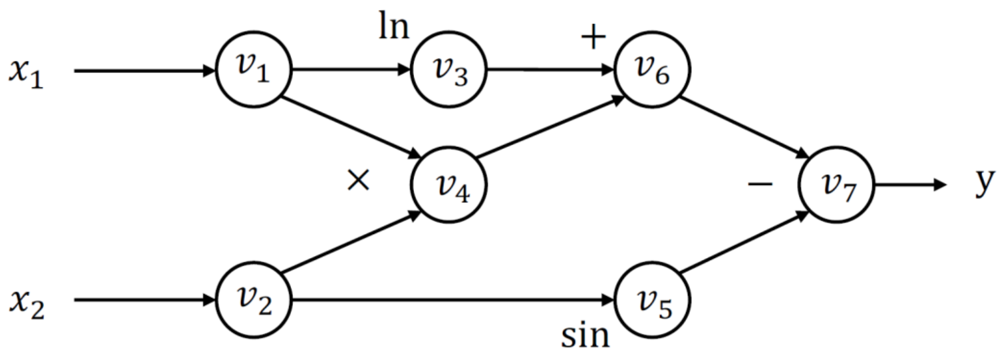
Symbolic Differentiation

- Use the model formula to derive gradients by sum, product and chain rules
  - $\frac{\partial\left(f(\theta)+g(\theta)\right)}{\partial(\theta)} = \frac{\partial f(\theta)}{\partial \theta} + \frac{\partial g(\theta)}{\partial \theta}$
  - $\frac{\partial\left(f(\theta)g(\theta)\right)}{\partial(\theta)} = g(\theta) \times \frac{\partial f(\theta)}{\partial \theta} + f(\theta) \times \frac{\partial g(\theta)}{\partial \theta}$
  - $\frac{\partial f(g(\theta))}{\partial(\theta)} = \frac{\partial f(g(\theta))}{\partial g(\theta)} \times \frac{\partial g(\theta)}{\partial \theta}$

$\Rightarrow$ Lots of repeated compute: $f(\theta) = \prod_{i=1}^{n} \theta_i$, $\frac{f(\theta)}{\partial \theta_k} = \prod_{j \neq k}^{n} \theta_j$

Recap: compute graph

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



- Each node represents an (intermediate) value in the computation. Edges present input/output relations.

Forward propagation steps

$v_1 = x_1 = 2$

$v_2 = x_2 = 5$

$v_3 =$

$v_4 =$

$v_5 =$

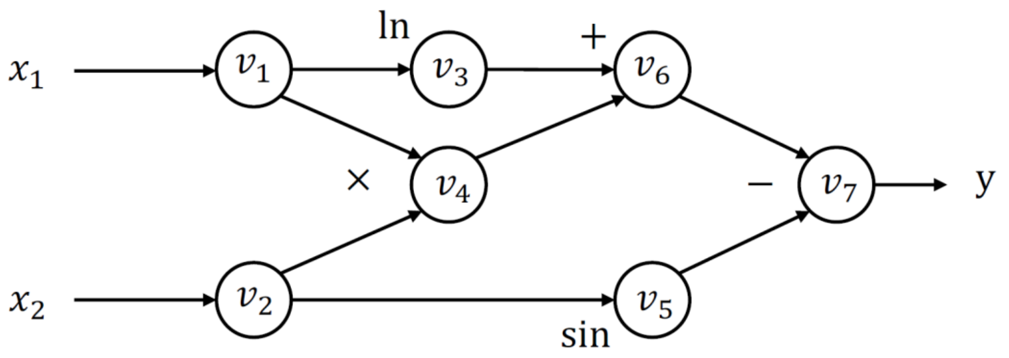
$v_6 =$

$v_7 =$

$y =$

Recap: compute graph

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



- Each node represents an (intermediate) value in the computation. Edges present input/output relations.

Forward propagation steps

$v_1 = x_1 = 2$

$v_2 = x_2 = 5$

$v_3 = \ln v_1 = \ln 2 = 0.692$

$v_4 = v_1 \times v_2 = 10$

$v_5 = \sin v_2 = \sin 5 = -0.959$

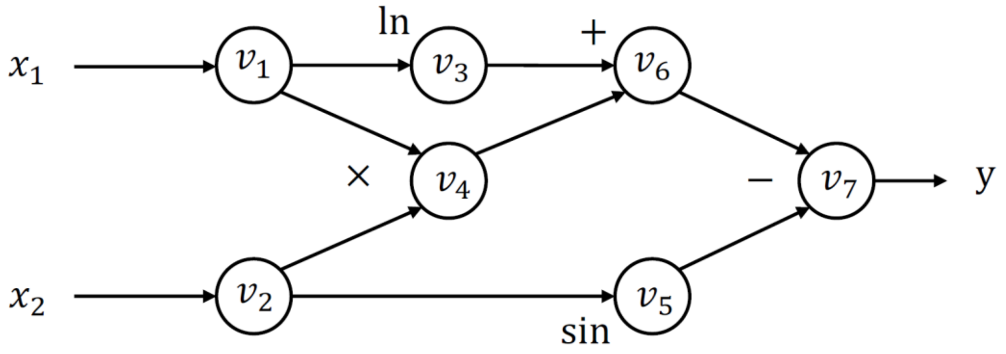
$v_6 = v_3 + v_4 = 10.693$

$v_7 = v_6 - v_5 = 11.652$

$y = v_7 = 11.652$

Forward Mode Automatic Differentiation (AutoDiff)

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



Forward propagation steps

$v_1 = x_1 = 2$
$v_2 = x_2 = 5$
$v_3 = \ln v_1 = \ln 2 = 0.692$
$v_4 = v_1 \times v_2 = 10$
$v_5 = \sin v_2 = \sin 5 = -0.959$
$v_6 = v_3 + v_4 = 10.693$
$v_7 = v_6 - v_5 = 11.652$
$y = v_7 = 11.652$

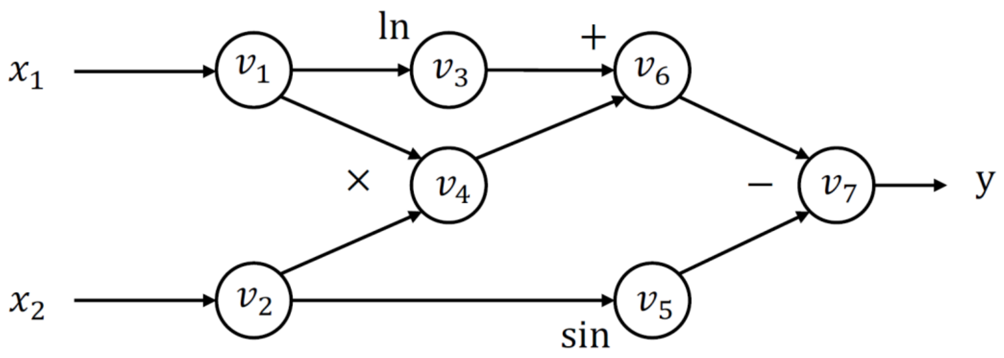
- Tweak the input and watch how the output changes

- **How much do you have?**
- **$100?**
- **$50?**

- **Guess**
- **Too much**
- **Too few**

Forward Mode Automatic Differentiation (AutoDiff)

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



Forward propagation steps

$v_1 = x_1 = 2$

$v_2 = x_2 = 5$

$v_3 = \ln v_1 = \ln 2 = 0.692$

$v_4 = v_1 \times v_2 = 10$

$v_5 = \sin v_2 = \sin 5 = -0.959$

$v_6 = v_3 + v_4 = 10.693$

$v_7 = v_6 - v_5 = 11.652$

$y = v_7 = 11.652$

Let $\Delta v_i = \frac{\partial v_i}{\partial x_1}$,

we can compute Δv_i by tweaking the inputs and perform forward propagation:

$\Delta v_1 = 1$

$\Delta v_2 = 0$

$\Delta v_3 =$

$\Delta v_4 =$

$\Delta v_5 =$

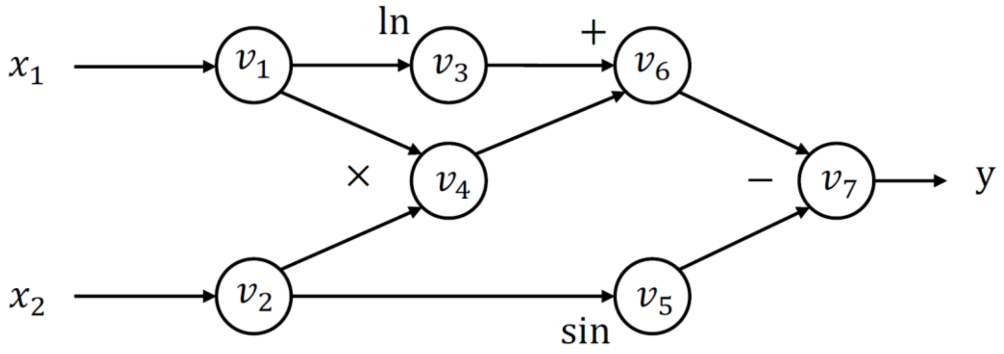
$\Delta v_6 =$

$\Delta v_7 =$

$\frac{\partial y}{\partial x_1} =$

Forward Mode Automatic Differentiation (AutoDiff)

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



Forward propagation steps

$v_1 = x_1 = 2$

$v_2 = x_2 = 5$

$v_3 = \ln v_1 = \ln 2 = 0.692$

$v_4 = v_1 \times v_2 = 10$

$v_5 = \sin v_2 = \sin 5 = -0.959$

$v_6 = v_3 + v_4 = 10.693$

$v_7 = v_6 - v_5 = 11.652$

$y = v_7 = 11.652$

Let $\Delta v_i = \frac{\partial v_i}{\partial x_1}$,

we can compute Δv_i by tweaking the inputs and perform forward propagation:

$\Delta v_1 = 1$

$\Delta v_2 = 0$

$$\Delta v_3 = \frac{\Delta v_1}{v_1} = 0.5$$

$\Delta v_4 = \Delta v_1 v_2 + \Delta v_2 v_1 = 1 \times 5 + 0 \times 2 = 5$

$\Delta v_5 = \Delta v_2 \cos v_2 = 0 \times \cos 5 = 0$

$\Delta v_6 = \Delta v_3 + \Delta v_4 = 0.5 + 5 = 5.5$

$\Delta v_7 = \Delta v_6 - \Delta v_5 = 5.5 - 0 = 5.5$

$$\frac{\partial y}{\partial x_1} = \Delta v_7 = 5.5$$

Forward Mode Automatic Differentiation (AutoDiff)

- However, each input x_i needs a whole forward propagation

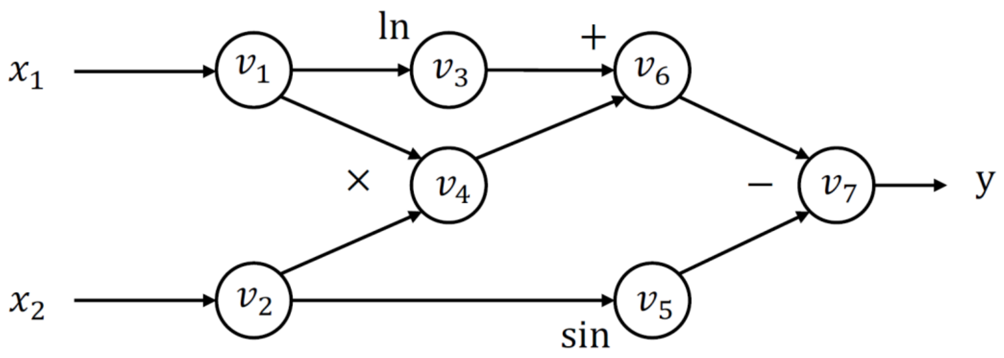
  $\Rightarrow$Very expensive

  $\Rightarrow$Hard to set proper Δx, know Δy only

  $\Rightarrow$Often used to check the correctness of coding

Reverse Mode Automatic Differentiation (AutoDiff)

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



Forward propagation steps

$v_1 = x_1 = 2$

$v_2 = x_2 = 5$

$v_3 = \ln v_1 = \ln 2 = 0.692$

$v_4 = v_1 \times v_2 = 10$

$v_5 = \sin v_2 = \sin 5 = -0.959$

$v_6 = v_3 + v_4 = 10.693$

$v_7 = v_6 - v_5 = 11.652$

$y = v_7 = 11.652$

Let $\Delta v_i = \frac{\partial y}{\partial v_i}$,

we can compute Δv_i in a reverse order of the graph

$$\Delta v_7 = \frac{\partial y}{\partial v_7} = 1$$

$\Delta v_6 =$

$\Delta v_5 =$

$\Delta v_4 =$

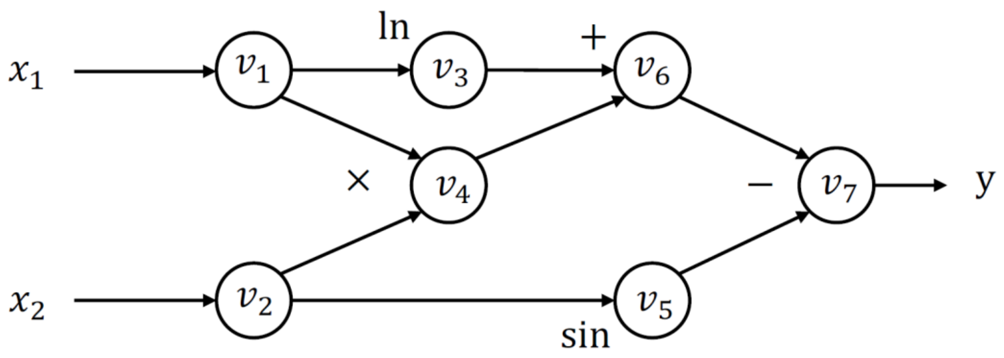
$\Delta v_3 =$

$\Delta v_2 =$

$\Delta v_1 =$

Reverse Mode Automatic Differentiation (AutoDiff)

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



Forward propagation steps

$v_1 = x_1 = 2$

$v_2 = x_2 = 5$

$v_3 = \ln v_1 = \ln 2 = 0.692$

$v_4 = v_1 \times v_2 = 10$

$v_5 = \sin v_2 = \sin 5 = -0.959$

$v_6 = v_3 + v_4 = 10.693$

$v_7 = v_6 - v_5 = 11.652$

$y = v_7 = 11.652$

Let $\Delta v_i = \frac{\partial y}{\partial v_i}$,

we can compute Δv_i in a reverse order of the graph

$$\Delta v_7 = \frac{\partial y}{\partial v_7} = 1$$

$$\Delta v_6 = \Delta v_7 \times \frac{\partial v_7}{\partial v_6} = \Delta v_7 \times 1 = 1$$

$$\Delta v_5 = \Delta v_7 \times \frac{\partial v_7}{\partial v_5} = \Delta v_7 \times (-1) = -1$$

$$\Delta v_4 = \Delta v_6 \times \frac{\partial v_6}{\partial v_4} = \Delta v_6 \times 1 = 1$$

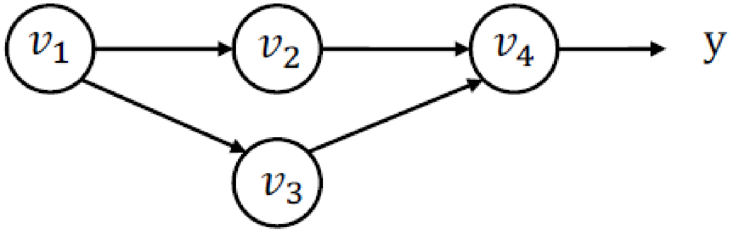
$$\Delta v_3 = \Delta v_6 \times \frac{\partial v_6}{\partial v_3} = \Delta v_6 \times 1 = 1$$

$$\Delta v_2 = \Delta v_5 \times \frac{\partial v_5}{\partial v_2} + \Delta v_4 \times \frac{\partial v_4}{\partial v_2} = \Delta v_5 \times \cos v_2 + \Delta v_4 \times v_1 = -\cos 5 + 2$$

$$\Delta v_1 = \Delta v_4 \times \frac{\partial v_4}{\partial v_1} + \Delta v_3 \times \frac{\partial v_3}{\partial v_1} = \Delta v_4 \times v_2 + \Delta v_3 \times \frac{1}{v_1} = 5 + \frac{1}{2} = 5.5$$

Derivation for branches

- In reverse model AutoDiff, gradients are summed up from branches



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

- Define partial adjoint $\Delta v_{i \to j} = \Delta v_j \times \frac{\partial v_j}{\partial v_i}$ for each pair of adjacent node i, j

- Then for a node with multiple outbound pathways,

$$\Delta v_i = \sum_{j \in \mathrm{adj}(i)} \Delta v_{i \to j}$$

We can compute partial adjoints, and then sum them together.

Reverse Model AutoDiff Algorithm

```
def gradient(out):
  node_to_grad = {out:  [1]}
  for i in reverse_topo_order(out):
```

$\Delta v_i = \sum_j \Delta v_{i\to j}$ = sum(node_to_grad[i])

for $k \in inputs(i)$:

compute $\Delta v_{k\to i} = \Delta v_i \dfrac{\partial v_i}{\partial v_k}$

append $\Delta v_{k\to i}$ to node_to_grad[k]

return adjoint of input Δv_{input}

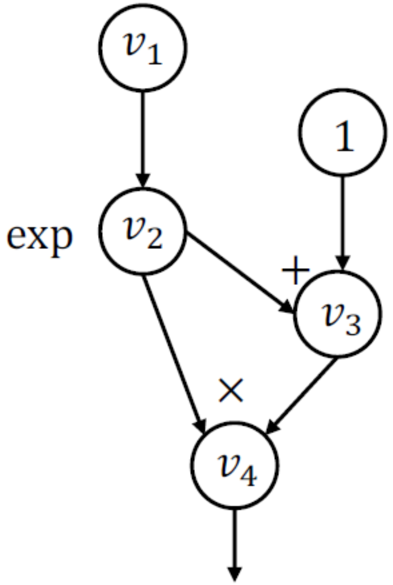
Reverse Model AutoDiff Algorithm

def gradient(out):
  node_to_grad = {out: [1]} **out:** dictionary to record a list of partial adjoints for each node
  for i in reverse_topo_order(out):
    $\bar{v}_i = \sum_j \bar{v}_{i\to j}$ = sum(node_to_grad[i])
    for $k \in inputs(i)$:
      compute $\bar{v}_{k\to i} = \bar{v}_i \dfrac{\partial v_i}{\partial v_k}$
      append $\bar{v}_{k\to i}$ to node_to_grad[k] Propagates partial adjoint to its input node
    return adjoint of input $\bar{v}_{input}$

Key is to compute the adjoint values for each node and construct the graph on the fly.

Reverse Model AutoDiff Algorithm

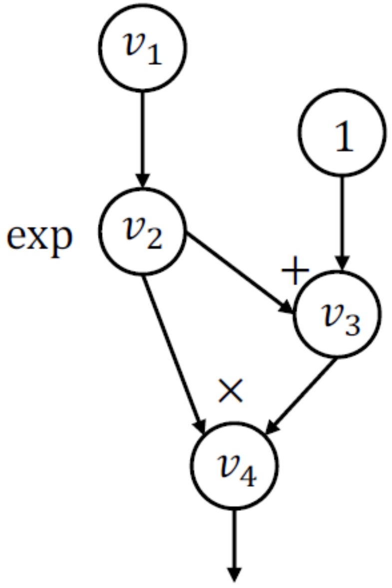
```
def gradient(out):
  node_to_grad = {out:  [1]}
  for i in reverse_topo_order(out):
    Δv_i = Σ_j Δv_{i→j} = sum(node_to_grad[i])
    for k ∈ inputs(i):
        compute Δv_{k→i} = Δv_i ∂v_i/∂v_k
        append Δv_{k→i} to node_to_grad[k]
    return adjoint of input Δv_input
```

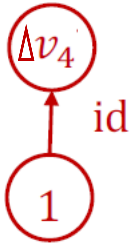


Reverse Model AutoDiff Algorithm

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

$i = 4$
node_to_grad: {
  4: [Δv_4]
}



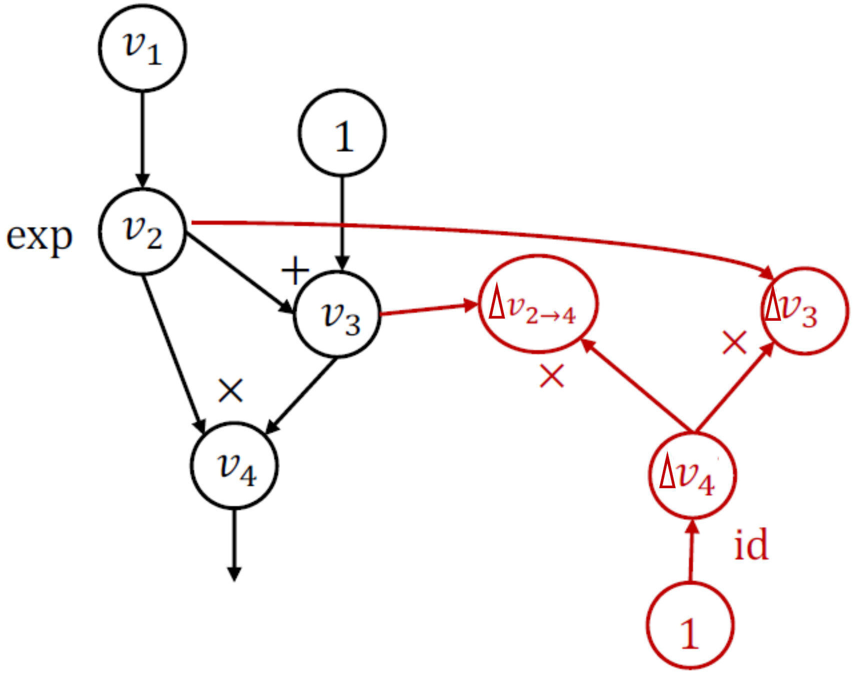


id = identity function

Reverse Model AutoDiff Algorithm

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

$i = 4$
node_to_grad: {
  2: [$\Delta v_{2\to4}$]
  3: [Δv_3]
  4: [Δv_4]
}

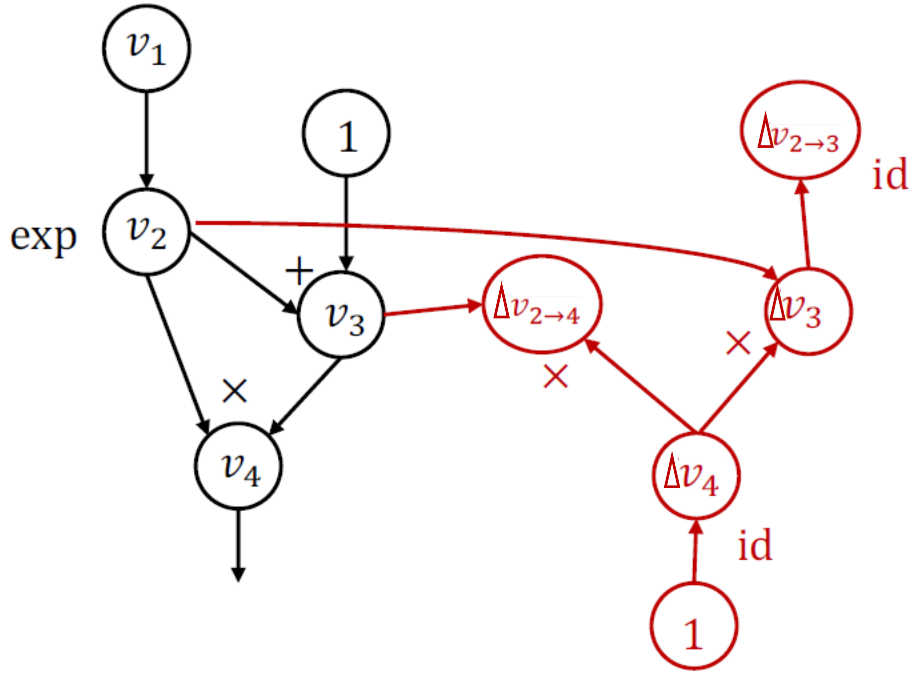


id = identity function

Reverse Model AutoDiff Algorithm

def gradient(out):
  node_to_grad = {out: [1]}
  for i in reverse_topo_order(out):
    $\bar{v}_i = \sum_j \bar{v}_{i\to j}$ = sum(node_to_grad[i])
    for $k \in inputs(i)$:
      compute $\bar{v}_{k\to i} = \bar{v}_i \dfrac{\partial v_i}{\partial v_k}$
      append $\bar{v}_{k\to i}$ to node_to_grad[k]
  return adjoint of input $\bar{v}_{input}$

$i = 3$
node_to_grad: {
  2: [$\bar{v}_{2\to 4}$, $\bar{v}_{2\to 3}$]
  3: [$\bar{v}_3$]
  4: [$\bar{v}_4$]
}



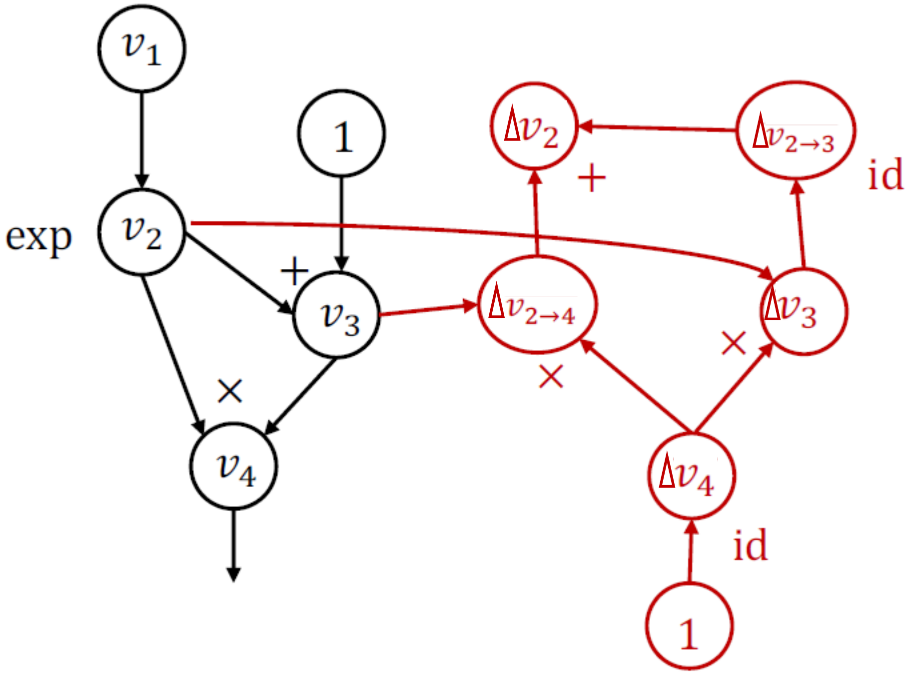
id = identity function

Reverse Model AutoDiff Algorithm

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

$i = 3$

```
node_to_grad: {
  2: [Δv_{2→4}, Δv_{2→3}]
  3: [Δv_3]
  4: [Δv_4]
}
```

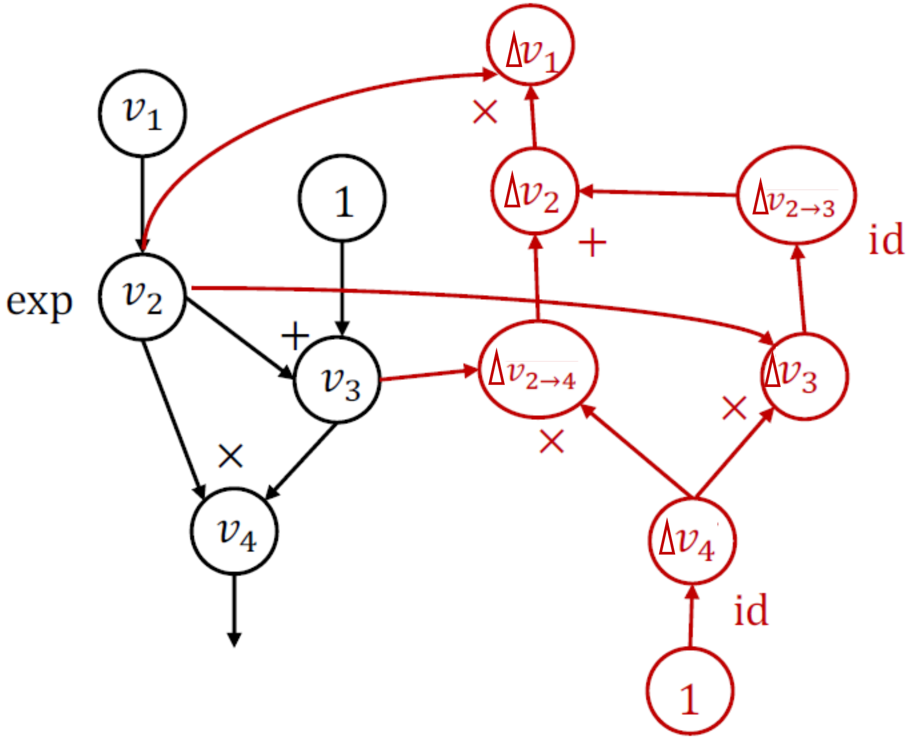


id = identity function

Reverse Model AutoDiff Algorithm

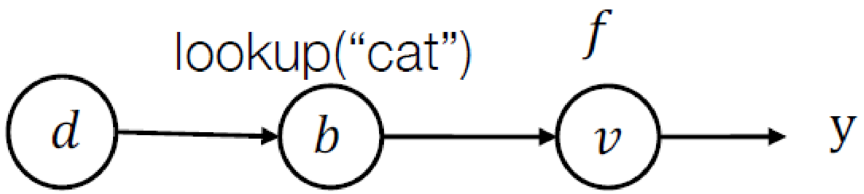
def gradient(out):
  node_to_grad = {out: [1]}
  for i in reverse_topo_order(out):
    $\Delta v_i = \sum_j \Delta v_{i\to j}$ = sum(node_to_grad[i])
    for $k \in inputs(i)$:
      compute $\Delta v_{k\to i} = \Delta v_i \dfrac{\partial v_i}{\partial v_k}$
      append $\Delta v_{k\to i}$ to node_to_grad[k]
  return adjoint of input Δv_{input}

$i = 2$
node_to_grad: {
  1: [Δv_1]
  2: [$\Delta v_{2\to4}$, $\Delta v_{2\to3}$]
  3: [Δv_3]
  4: [Δv_4]
}



id = identity function

Reverse Model AutoDiff on discrete data structures



Define adjoint data structure

$$\Delta d = \left\{ "\mathrm{cat}": \frac{\partial y}{\partial a_0}, "\mathrm{dog}": \frac{\partial y}{\partial a_1} \right\}$$

Forward propagation:

$$d = \{"\mathrm{cat}": a_0, "\mathrm{dog}": a_1\}$$

$$b = d["\mathrm{cat}"]$$

$$v = f(b)$$

Reverse AutoDiff:

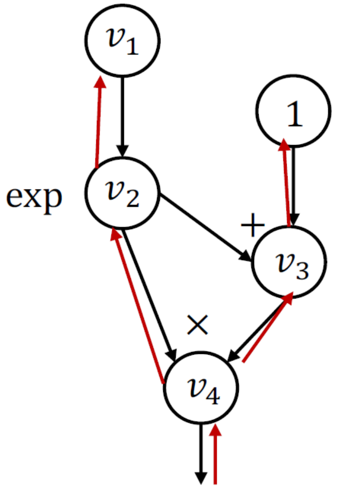
$$\Delta b = \frac{\partial v}{\partial b} \times \Delta v$$

$$\Delta d = \{"\mathrm{cat}": \Delta b\}$$

- Define adjoint values in the same way as in the forward propagation.

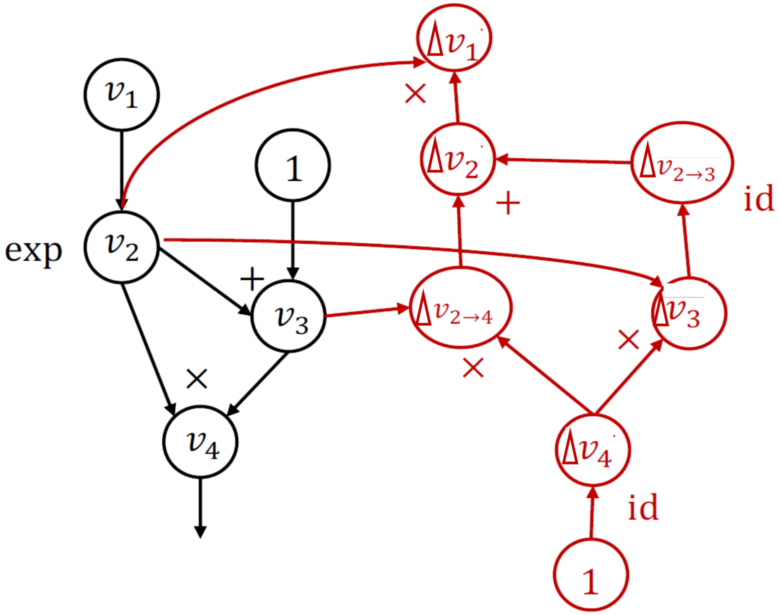
Compute in-place vs. Reverse Model AutoDiff

Compute in-place



- Run backprop on the forward graph
- Used in earlier frameworks (caffe etc.)

Reverse mode AutoDiff w/ compute graph



- Construct separate graph nodes for adjoints
- Used in modern frameworks (Pytorch etc.)

Ways to compute gradients

	Pros	Cons
Numerical differentiation	Intuitive & easy to compute	Numerical error
Symbolic differentiation		Repeated compute
Forward model AutoDiff		Repeated compute
Backward model AutoDiff	Scalable & saves compute	Memory consumption