Auto-Differentiation

By Rajiv Koliwad, Sasha Arditi , and Malik Kurtz

Contents

Why Auto-Differentiation?

- There are several methods of computing derivatives
  - Symbolic Differentiation: Uses algebraic manipulation to calculate exact derivatives (exact but leads to expression swell)
  - Numerical Differentiation: Uses the method of finite differences to approximate derivatives (simple to implement but can lead to accumulating rounding errors and requires many function evaluations)
  - Manual Differentiation: Derivatives manually derived and implemented by the user (simple but not flexible, not dynamic since hard-coded)
- Auto-Differentiation combines exact precision and computational efficiency by systematically applying the chain rule for functions made up of arbitrary code
- Primary method used for deriving gradients in popular deep-learning libraries like PyTorch and TensorFlow

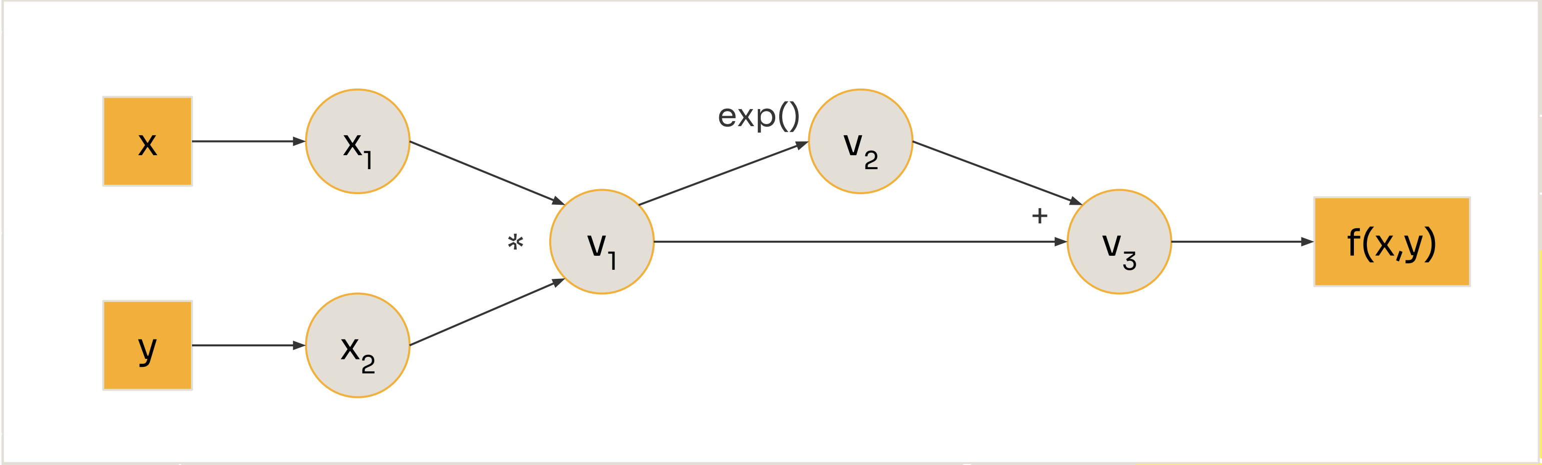
What is Auto-Differentiation?

- Auto-Differentiation is a technique used to calculate the Jacobian Matrix of a function with n inputs and m outputs
    - If we have a cost function with n inputs (parameters) and a single output, as is the case in a regression task, then the Jacobian matrix produced by Auto-Differentiation will be a 1 x n matrix of partial derivatives which is just the typical gradient vector
- There are two types, Forward Mode accumulation and Reverse Mode accumulation.
    - Forward Mode accumulation builds the Jacobian matrix one column at a time, optimal when there are fewer inputs
    - Reverse Mode accumulation builds the Jacobian matrix one row at a time, optimal when there are fewer outputs (most common in machine learning problems)
- Reverse Mode accumulation uses a computational graph to record intermediate operations during the forward pass of the function and reuses them during the backward pass to efficiently calculate partial derivatives

Example?

- The next few slides will feature a simple example of reverse accumulation.
- Because we have limited time, this example will not feature complex matrices like backpropagation for a neural network would: rather, we will showcase reverse accumulation on a fairly simple function with only two variables.

Let us attempt to evaluate the function $f(x,y)=xy+e^{xy}$ when $x=1,y=2$



note: this is not referencing our original xy, but the directly preceding argument(s)

First, generate forward trace

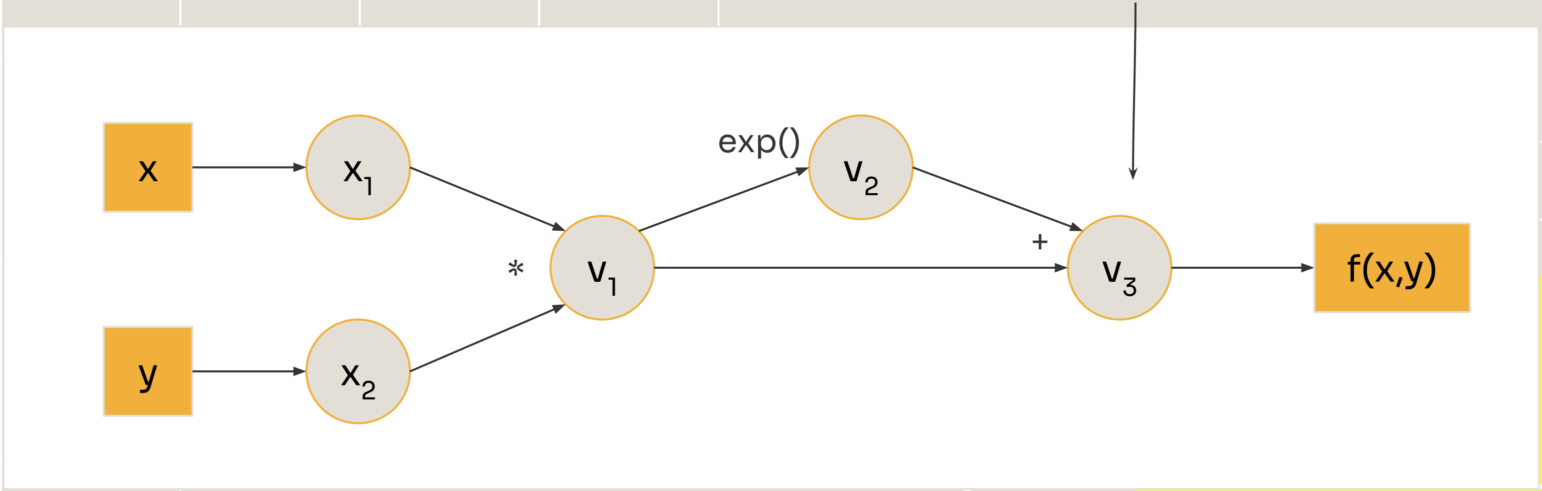
This is similar to what we would do in forward accumulation, but rather than actually calculate with the chain rule, we only store information at each step in terms of the step immediately preceding it

trace	function	value (x=1, y=2)	derivative of 1st argument	derivative of 2nd argument	values of derivatives
x_1	x_1	1	1	0	[1,0]
x_2	x_2	2	0	1	[0,1]
v_1	x_1x_2	2	x_2	x_1	[2,1]
v_2	$\exp(v_1)$	e^2	$\exp(v_1)$	0 *(no 2nd arg)*	$[e^2,0]$
v_3	v_1+v_2	$2+e^2$	1	1	[1,1]

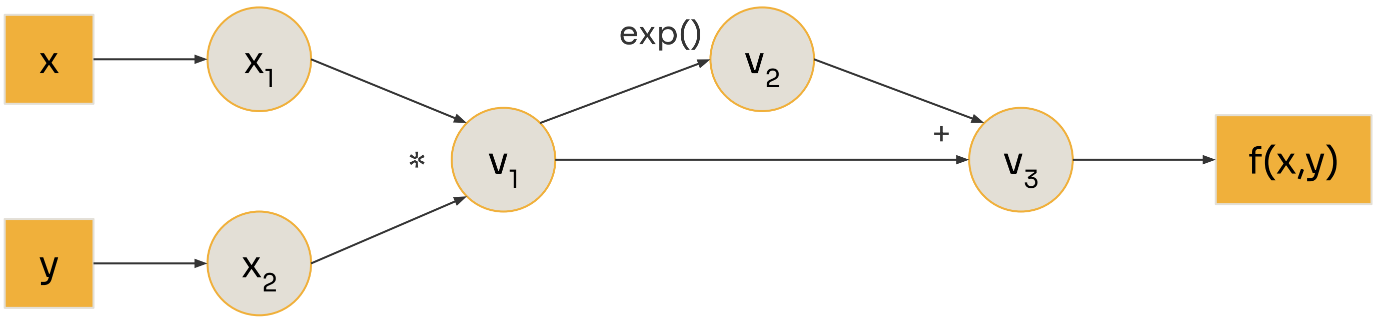
Then, begin reverse passes

We will set the partial derivative with respect to the final node as 1.

$v'_3 = df/dv^3 = 1$

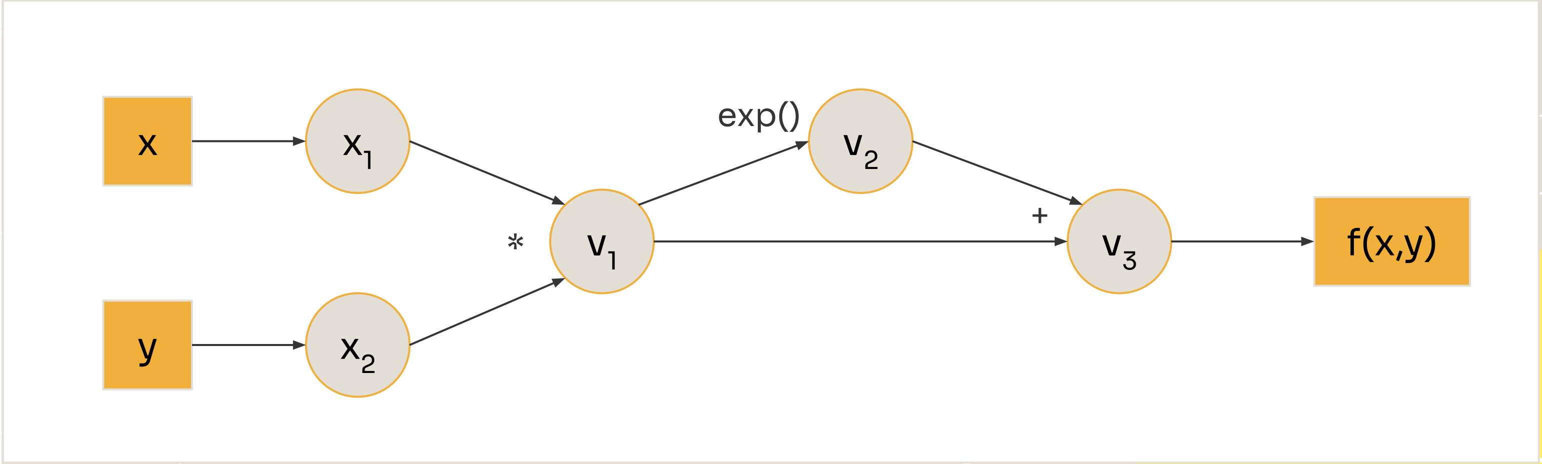


$$v'_2 = (df/dv_2) = (df/dv_3)(dv_3/dv_2) = 1 * 1 = 1$$



for v'_1 , we must sum the children:

$$v_1' = (df/dv_3)(dv_3/dv_1) + (df/dv_2)(dv_2/dv_1) = 1*1 + 1*e^2 = 1+e^2$$

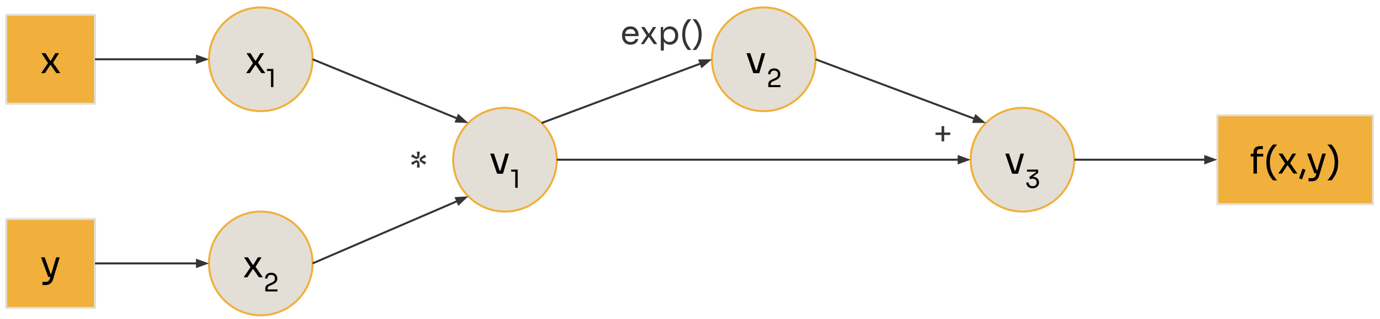


finally, we can calculate x_1 and x_2

$x_2' = (df/dv_1)(dv_1/dx_2) = (1+e^2)x_1 = 1 + e^2$

$x_1' = (df/dv_1)(dv_1/dx_1) = (1+e^2)x_2 = 2 + 2e^2$

Thus we have:
$df/dy = 1 + e^2$
$df/dx = 2 + 2e^2$



Implementation Steps

1.) Create “value” object to hold solved parts of equation and and associated derivative

2.) As more parts of the equation are solved, create “value” objects, linking parent objects with child objects

3.) Define functions like add and subtract in order both for calculating final value and for calculating derivative for “value” objects.

4.) Parse “value” tree, calculating derivatives of output with respect to input values

Benchmarking

Finally, we can benchmark our software.

Create small neural network for classification

Compare neural network performance with our auto differentiation package compared with industry standard.

Can compare model performance/training loss on same data

Thank you