Automatic differentiation

COMS 4771 Fall 2025

Primary "technical work" in implementing gradient descent method:
Derive formula and write code for gradient computation ∇J

- Like doing long division by hand (i.e., without electronic calculators)
- Fairly straightforward, but can be tedious and easy to make mistakes

Automatic differentiation (autodiff):

- Method for automatically computing derivatives of functions specified by straight-line programs
- Originally developed by Seppo Linnainmaa in his 1970 MSc thesis
- Gradient of a function can be computed this way in the roughly same amount of time it takes to compute the function itself (!)

Example: $J(w) = x^\intercal w$

- For each $j = 1, \ldots, d$, compute

$$\frac{\partial J}{\partial w_j}(w) = \underline{\hspace{2cm}}$$

- Time to compute function and gradient: __________

Example: $J(w) = g(f(w))$ where $f(w) = x^\intercal w$ and $g(t) = \text{logistic}(t)$

- For each $j = 1, \ldots, d$, compute

$$\frac{\partial J}{\partial w_j}(w) = \underline{\hspace{6cm}}$$

- Time to compute function: ________
- Time to compute gradient: naïvely $O(d^2)$, but easy to get $O(d)$

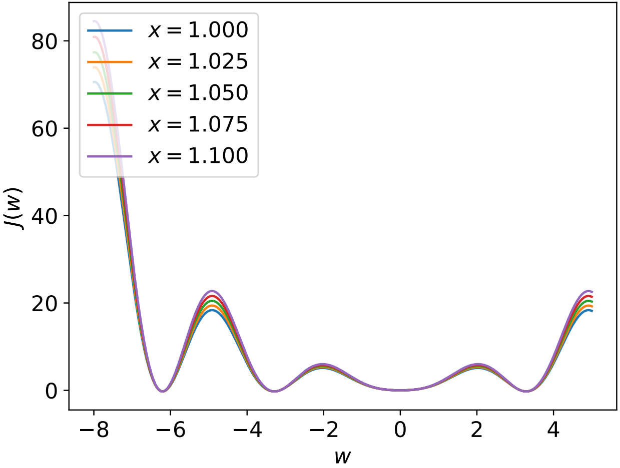
Example: tower of exponentials $J(w) = \exp(\exp(\exp(\cdots\exp(xw)\cdots)))$
(for scalar x and w)

We only want single number ($\frac{\partial J}{\partial w}$), but function is more complicated

$$\frac{\partial}{\partial w}\{e^{e^{e^{e^{e^{e^{e^{xw}}}}}}}\} = e^{e^{e^{e^{e^{e^{e^{xw}}}}}}}\, e^{e^{e^{e^{e^{e^{xw}}}}}}\, e^{e^{e^{e^{e^{xw}}}}}\, e^{e^{e^{e^{xw}}}}\, e^{e^{e^{xw}}}\, e^{e^{xw}}\, e^{xw} x$$

- Time to compute tower of exponentials of height h: ________
- Time to compute derivative: ______________________________

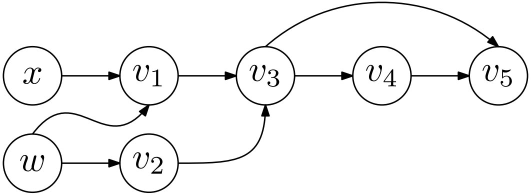
Example: $J(w) = xw\sin(w) + (xw\sin(w))^2$
(for scalar x and w)



Write J as a straight-line program: each line declares a new variable as a function of inputs (e.g., w), constants (e.g., x), or previously defined variables

$$J(w) = xw\sin(w) + (xw\sin(w))^2$$

$$\begin{aligned}
v_1 &:= \mathrm{prod}(x, w) \\
v_2 &:= \sin(w) \\
v_3 &:= \mathrm{prod}(v_1, v_2) \\
v_4 &:= \mathrm{square}(v_3) \\
v_5 &:= \mathrm{sum}(v_3, v_4)
\end{aligned}$$



Computation graph $G = (V, E)$

All functions used in straight-line program must come with subroutines for computing “local” partial derivative

Example:

$$
\begin{aligned}
v_3 &:= \operatorname{prod}(v_1, v_2) \\
\frac{\partial v_3}{\partial v_1} &= \frac{\partial \operatorname{prod}(v_1, v_2)}{\partial v_1} = v_2 \\
\frac{\partial v_3}{\partial v_2} &= \frac{\partial \operatorname{prod}(v_1, v_2)}{\partial v_2} = v_1
\end{aligned}
$$

Stage 1: **Forward pass**

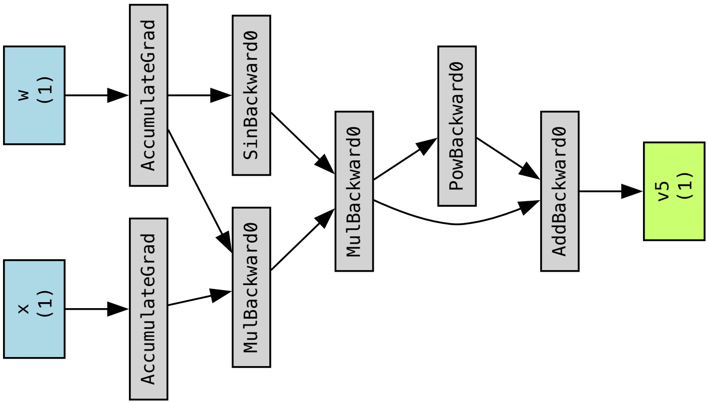
- Compute value of each node given inputs in a forward pass through the G (starting from inputs x and w)
- Save values at all intermediate nodes

Stage 2: **Backward pass**

- Compute partial derivative $\frac{\partial v_5}{\partial u}$ of output (v_5) with respect to each node variable u, evaluated at current node values
- Do this in reverse topological order; save intermediate results!

Chain rule: $$\frac{\partial v_5}{\partial u} = \sum_{(u,v) \in E} \frac{\partial v_5}{\partial v} \cdot \frac{\partial v}{\partial u}$$

- Time to compute function and partial derivatives: $O(|V| + |E|)$
- Modern numerical software facilitates construction of the straight-line program



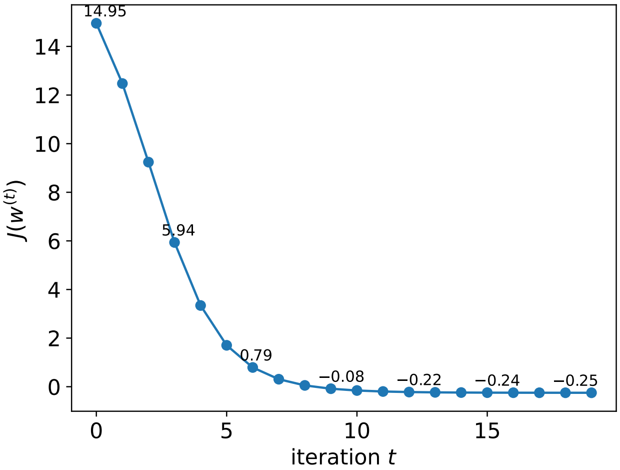
Setup

```
import torch

x = torch.Tensor([1])
w = torch.Tensor([-4.9])
w.requires_grad = True

def J(w):
  v1 = x * w
  v2 = torch.sin(w)
  v3 = v1 * v2
  v4 = torch.pow(v3, 2)
  v5 = v3 + v4
  return v5
```

Gradient descent code

```
for t in range(50):
  objective_value = J(w)
  objective_value.backward()
  with torch.no_grad():
    w -= 0.01 * w.grad
    w.grad.zero_()
```

Gradient descent on $J(w)$, starting from $w^{(0)} = -4.9$, using $\eta_t = 0.01$



Converges to $w \approx -3.294$, $J(w) = -2.5$, $\frac{\partial J}{\partial w}(w) \approx 0$