Jacobian-Vector & Vector-Jacobian products

- An intermezzo -

Recap on gradients

- For a scalar-valued function $f : \mathbb{R}^n \to \mathbb{R}$ the gradient is *another function* $\nabla f : \mathbb{R}^n \to \mathbb{R}^n$ defined in location $x = (x_1, \ldots, x_n), \nabla f(x) = [\partial f / \partial x_1, \ldots, \partial f / \partial x_n]^T$
- Jacobians generalise gradients for multi-dimensional functions $\dfrac{\partial f}{\partial x} : \mathbb{R}^n \to \mathbb{R}^m$
- It shows how much the function output changes for small perturbations in the respective input dimension
- The gradient is a *linear function* as every partial derivative is linear

$$\frac{\partial}{\partial x_i} f(x) = \lim_{\epsilon \to 0} \frac{f(x_1, \ldots, x_i + \epsilon/2, \ldots, x_n) - f(x_1, \ldots, x_i - \epsilon/2, \ldots, x_n)}{\epsilon}$$

Recap on backdrop

- For a function $f : \mathbb{R}^n \to \mathbb{R}$ comprising a cascade of modules $f = \ell \circ f_L \circ \cdots \circ f_1(x, \theta)$
- Backpropagation (aka auto-differentiation) is implements the chain rule

$$\frac{d\ell}{d\theta_l} = \frac{d\ell}{df_L} \cdot \frac{df_L}{df_{L-1}} \cdot \cdots \cdot \frac{df_l}{d\theta_l}, \quad \text{where } \frac{df_l}{df_{l-1}} \in \mathbb{R}^{n_l \times n_{l-1}}$$

- Forward-mode backprop (right to left), reverse-mode backprop (left to right)
- With forward-mode we multiply Jacobians from the right (Jacobian-vector products)
- With reverse-mode we multiply Jacobians from the left (vector-Jacobian products)

Jacobian-Vector & Vector-Jacobian products

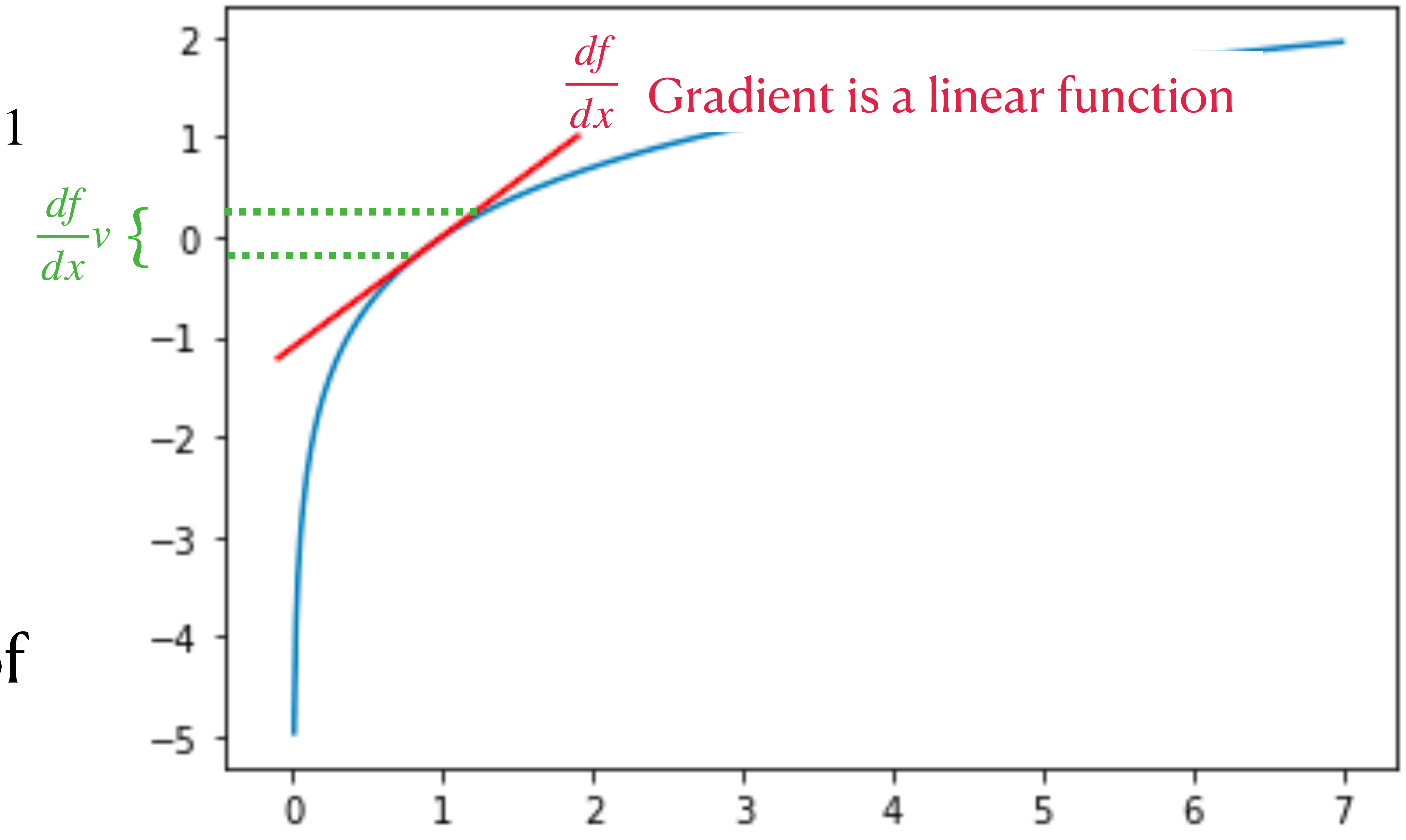
- Jacobians are everywhere in backdrop as they generalise gradients
- Often, in our auto-diff frameworks we must implement the Jacobian-vector (JVP) or vector-Jacobian (VJP) computations depending on whether the framework implements forward-mode or reverse-mode auto-differentiation
- Equivalent, but have different conceptual and computational characteristics

Jacobian-Vector & Vector-Jacobian products

- JVPs right-multiply: $\partial f(x)v, v \in \mathbb{R}^{n\times 1}$
- VJP left-multiply: $w^T \partial f(x), w \in \mathbb{R}^{m\times 1}$
- JVP/VJP capture 'total change' in function output when perturbed

$$\frac{\Delta f}{\Delta x} \approx \frac{df}{dx} \Rightarrow f_{new} \approx \frac{df}{dx}\Delta x + f_{old}$$

- By linear approximation (Jacobian) of how much function changes locally



Jacobian-Vector products (JVP)

- With JVP we mutliply from the right: we "weight-average" partial derivatives $\partial f_j / \partial x_i$ across all input dimensions to see how the j-th output dimension is affected
- In other words "how much each output dimension change for a small nudge to input"?

$$\partial f(x) v = \begin{bmatrix} \boxed{\frac{\partial f_1(x)}{\partial x_1} \quad \frac{\partial f_1(x)}{\partial x_2} \quad \cdots \quad \frac{\partial f_1(x)}{\partial x_n}} \\ \cdots \quad \cdots \quad \cdots \quad \cdots \\ \frac{\partial f_m(x)}{\partial x_1} \quad \frac{\partial f_m(x)}{\partial x_2} \quad \cdots \quad \frac{\partial f_m(x)}{\partial x_n} \end{bmatrix} \begin{bmatrix} \boxed{\begin{matrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{matrix}} \end{bmatrix} = \begin{bmatrix} \boxed{\sum_i \frac{\partial f_1(x)}{\partial x_i} v_i} \\ \sum_i \frac{\partial f_2(x)}{\partial x_i} v_i \\ \vdots \\ \sum_i \frac{\partial f_m(x)}{\partial x_i} v_i \end{bmatrix}$$

Vector-Jacobian products (VJP)

- With VJP we multiply from the left: it quantifies how much the specific i-th input dimension affected all outputs by "weight-averaging" across all output dimensions

$$w^T \partial f(x) = \boxed{[w_1, w_2, \cdots, w_m]} \begin{bmatrix} \frac{\partial f_1(x)}{\partial x_1} & \frac{\partial f_1(x)}{\partial x_2} & \cdots & \frac{\partial f_1(x)}{\partial x_n} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial f_m(x)}{\partial x_1} & \frac{\partial f_m(x)}{\partial x_2} & \cdots & \frac{\partial f_m(x)}{\partial x_n} \end{bmatrix}$$

$$= \left[\boxed{\sum_i \frac{\partial f_i(x)}{\partial x_1} w_i,} \sum_i \frac{\partial f_i(x)}{\partial x_2} w_i, \cdots, \sum_i \frac{\partial f_i(x)}{\partial x_n} w_i, \right]$$

Interpreting VJPs

- Assume we want to find out *how much* our loss gradient ($\Delta\ell$) is affected by an output perturbation Δy

$$\Delta\ell = \frac{\mathrm{d}\ell}{\mathrm{d}y}^{\top} \Delta y = l^{\top}\Delta y$$

which itself is the result of perturbing the input by Δx, that is $\Delta y = \dfrac{\partial f}{\partial x}\,\Delta x$

$$\Delta\ell = l^{\top}\Delta y = \underbrace{l^{T}\frac{\partial f}{\partial x}}_{\text{VJP}}\,\Delta x = \lambda^{T}\Delta x$$

- For a <u>linear</u> function l representing how the loss changes ($\Delta\ell$) w.r.t. nudges to its direct input Δy
- VJP represents a corresponding <u>linear</u> function λ causing same change w.r.t. nudges to indirect input Δx

Computations in JVP & VJP

- JVPs correspond to forward-mode auto-differentiation
- VJP correspond to reverse-mode auto-differentiation
- Since the loss function is scalar ($m = 1$) and inputs n often in millions, VJP makes more compact computations compared to JVP

$$\underbrace{\frac{\partial \ell}{\partial y}}_{1\times 10} \cdot \underbrace{\frac{\partial y}{\partial z}}_{10\times 100} \cdot \underbrace{\frac{\partial z}{\partial x}}_{100\times 100{,}000}$$

- VJP is more popular in auto-diff libraries

JVPs & VJPs on our fixed points

Fixed-point JVP

- Reminder: the gradient of our fixed point

$$\partial_x z^\star(x) = \left[I - \partial_{z^\star} f(x, z^\star(x)) \right]^{-1} \partial_x f(x, z^\star)$$

- How does the fixed-point gradient change with nudging on the right

$$\partial_x z^\star(x) v = \left[I - \partial_{z^\star} f(x, z^\star(x)) \right]^{-1} \partial_x f(x, z^\star) v$$

Fixed-point JVP

- We first compute the JVP: $\partial_x z^\star(x) v = \left[I - \partial_{z^\star} f(x, z^\star(x)) \right]^{-1} \underbrace{\partial_x f(x, z^\star) v}_{u}$

- For the rest, even if the inverse is too hard to compute, we can do:

$$w = \left[I - \partial_{z^\star} f(x, z^\star(x)) \right]^{-1} u \Rightarrow \qquad \left[I - \partial_{z^\star} f(x, z^\star(x)) \right] w = u \Rightarrow$$

$$w = u + \partial_{z^\star} f(x, z^\star(x)) w = \partial_x z^\star(x) v$$

- We find how fixed-point changes, $\partial_x z^\star(x) v$, by another fixed-point problem

Fixed point VJP

- Similarly, we can nudge from the left: $l^T \partial_x z^\star(x) = l^T \left[I - \partial_{z^\star} f(x, z^\star(x))\right]^{-1} \partial_x f(x, z^\star)$
- For $u^T = l^T \left[I - \partial_{z^\star} f(x, z^\star(x))\right]^{-1}$ we have

$$u^T = l^T + u^T \partial_{z^\star} f(x, z^\star(x))$$

- So, we first compute another fixed point u with a fixed-point solver
- Then, the change to our fixed point is another VJP, $u^T \partial_x f(x, z^\star)$

Pros and cons of implicit differentiation

- General auto-diff works, but it is memory expensive, often computationally expensive, and numerically unstable
- For implicit differentiation we just need the final fixed-point for the back-propagation, which we can get with any fixed-point solver
- We do not care for all the intermediate solution points of the fixed-point solver
- Intuitively, implicit differentiation follows the logic "1. linearise around the fixed point, 2. then solve the linear system"
- The "2. then solve the linear system" can be done again with another fixed-point solver but we are free to choose