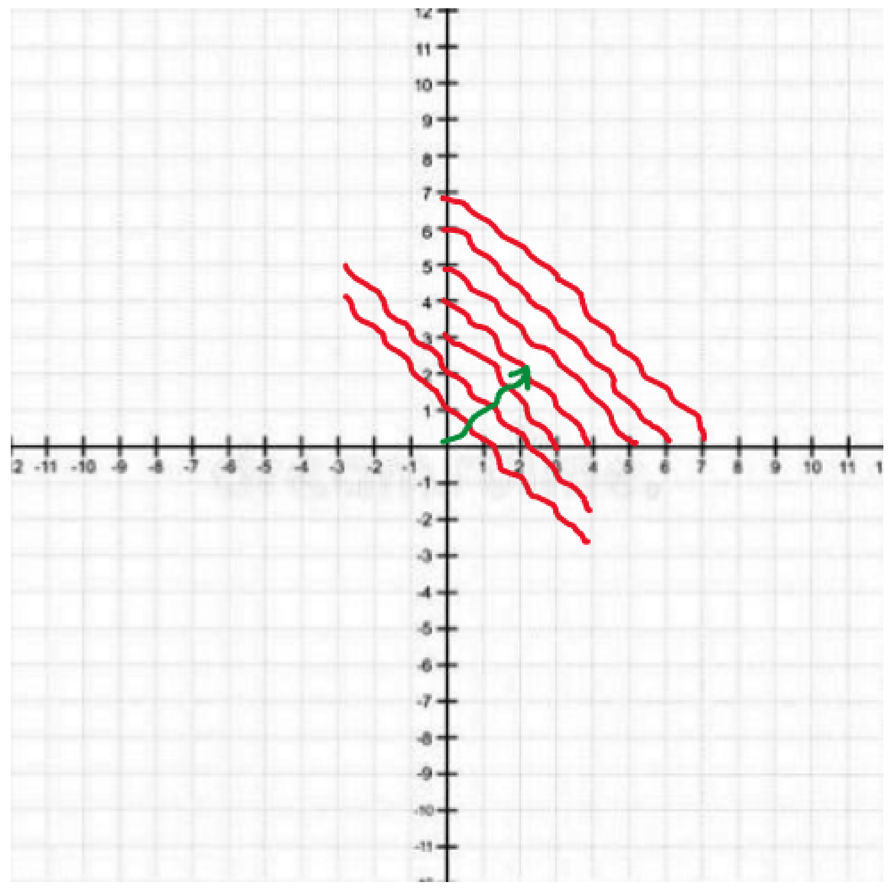
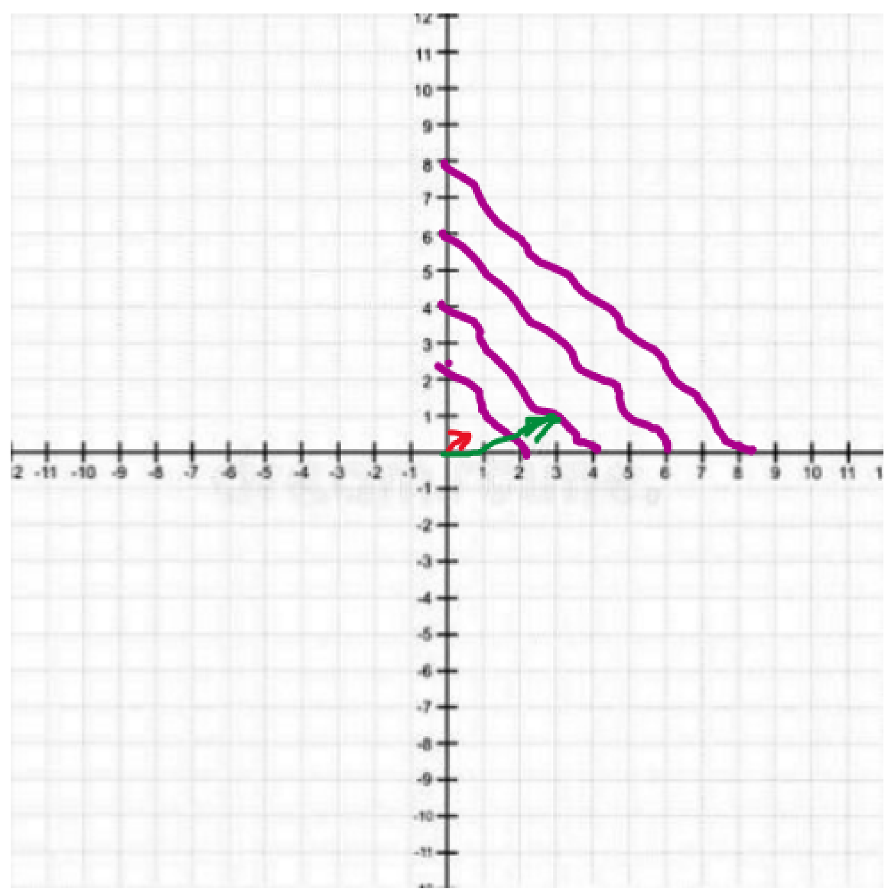
Derivative as a Convector

Monday, May 2, 2022 9:11 PM



$$\begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 2 \end{bmatrix} = 4$$

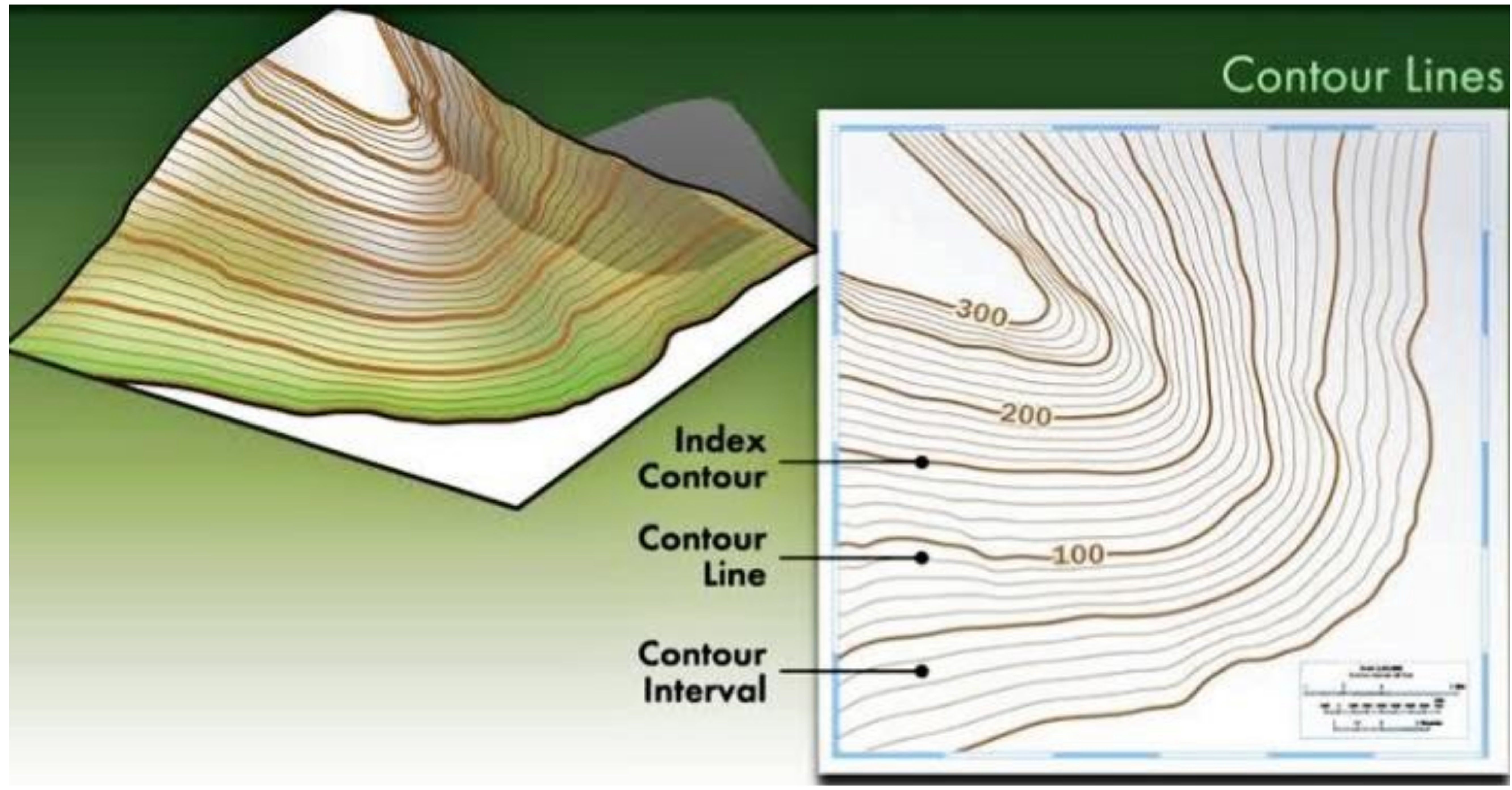


$$f(x_1, x_2) = \tfrac{1}{2} x_1 + \tfrac{1}{2} x_2$$

$$\nabla_x f = \begin{bmatrix} \tfrac{1}{2} & \tfrac{1}{2} \end{bmatrix}$$

$$\tilde{y} = \begin{bmatrix} \tfrac{1}{2} & \tfrac{1}{2} \end{bmatrix} \begin{bmatrix} 1 \\ 3 \end{bmatrix} \qquad \tilde{y} = \frac{dy}{dx_1} \cdot x_1 + \frac{dy}{dx_2} x_2$$

$$\tilde{y} = 2$$



$$f(x) = \tilde{y} = \nabla_x f \cdot x$$

$$= \begin{bmatrix} \frac{d}{dx_1} y & \frac{d}{dx_2} y \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$\|\tilde{y}\|_2^2 = \left\| \begin{bmatrix} \frac{d}{dx_1} y & \frac{d}{dx_2} y \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \right\|_2^2 \qquad \left(\begin{matrix} (xy)^T = y^T x^T \\ \text{and} \\ \|x\|_2 = \sqrt{x^T \cdot x} \end{matrix} \right)$$

$$= \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} \frac{d}{dx_1} y \\ \frac{d}{dx_2} y \end{bmatrix} \begin{bmatrix} \frac{d}{dx_1} y & \frac{d}{dx_2} y \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$= \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} \frac{d}{dx_1} y \frac{d}{dx_1} y & \frac{d}{dx_1} y \frac{d}{dx_2} y \\ \frac{d}{dx_2} y \frac{d}{dx_1} y & \frac{d}{dx_1} y \frac{d}{dx_2} y \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

↳ Riemannien Metric Tensor
For NN's its (w.r.t. loss)
"Fisher Information Matrix"

Vector Calculus 2 of 2

Backpropagation and Automatic Differentiation

The use of gradients in neural networks is fundamental to their effectiveness. Specifically,

- We use the gradient of an error function with respect to the model's parameters as a means of figuring out how to best update the model parameter to reduce the error.
- Given the core nature of this mechanism it is important to
  - Avoid the need for manual derivative calculation in larger models
    - ⋆ Time consuming and error prone
  - Perform the automatic differentiation in an efficient way.

Gradients in a Deep Network

Most of deep learning relies on many-level function composition

$$\mathbf{y} = (\mathbf{f}_K \circ \mathbf{f}_{K-1} \circ \cdots \circ \mathbf{f}_1)(\mathbf{x}) \tag{17}$$

$$= \mathbf{f}_K(\mathbf{f}_{K-1}(\ldots(\mathbf{f}_1(\mathbf{x}))\ldots)) \tag{18}$$

where $\mathbf{x}$ are the inputs, $\mathbf{y}$ are the observations (e.g class labels)

- every function $\mathbf{f}_i$, $i = 1, \ldots, K$, possesses its own parameters.

Manually finding the derivative of $\mathbf{f}_K$ with respect to one of the parameter sets deep within the computational layering quickly becomes intractable.

Gradients in a Deep Network

In a multi layer neural network we have that, in the ith layer,

$$\mathbf{f}_i(\mathbf{x}_{i-1}) = \sigma\left(\mathbf{A}_{i-1}\mathbf{x}_{i-1} + \mathbf{b}_{i-1}\right) \tag{19}$$

- The $\mathbf{x}_{i-1}$ is the output of the $i-1$ layer.
- The σ is and an activation function.
  - Common ones are sigmoid, tanh, and ReLU
- Both $\mathbf{A}_{i-1}$ and $\mathbf{b}_{i-1}$ are our model parameter from this layer.

Gradients in a Deep Network

We can consider our neural network as

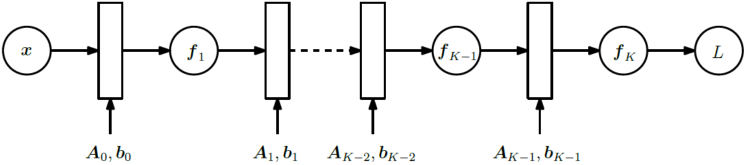
$$\mathbf{f}_0 := \mathbf{x} \tag{20}$$

$$\mathbf{f}_i := \sigma\left(\mathbf{A}_{i-1}\mathbf{f}_{i-1} + \mathbf{b}_{i-1}\right), \ i = 1, \ldots, k \tag{21}$$

where we want to minimize the squared loss

$$L(\boldsymbol{\theta}) = \|\mathbf{y} - \mathbf{f}_k(\boldsymbol{\theta}, \mathbf{x})\|_2 \tag{22}$$

By changing the model parameters $\boldsymbol{\theta} = \{\mathbf{A}_0, \mathbf{b}_0, \ldots, \mathbf{A}_{k-1}, \mathbf{b}_{k-1}\}$



Source: M.P. Deisenroth *et al*, Mathematics for Machine Learning (First Edition)

Gradients in a Deep Network

To obtain the gradients with respect to the parameter set $\boldsymbol{\theta}$,

- we require the partial derivatives of L with respect to the parameters $\boldsymbol{\theta}_j = \{\mathbf{A}_j, \mathbf{b}_j\}$ of each layer.
- The chain rule allows us to determine the partial derivatives as

$$\frac{\partial L}{\partial \boldsymbol{\theta}_{k-1}} = \frac{\partial L}{\partial \mathbf{f}_k} \frac{\partial \mathbf{f}_k}{\partial \boldsymbol{\theta}_{k-1}} \tag{23}$$

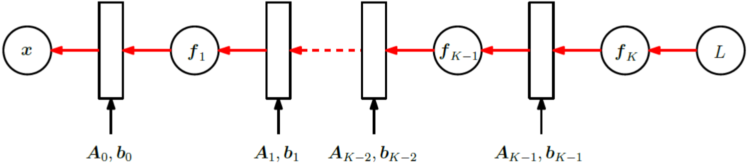
$$\frac{\partial L}{\partial \boldsymbol{\theta}_{k-2}} = \frac{\partial L}{\partial \mathbf{f}_k} \boxed{\frac{\partial \mathbf{f}_k}{\partial \mathbf{f}_{k-1}} \frac{\partial \mathbf{f}_{k-1}}{\partial \boldsymbol{\theta}_{k-2}}} \tag{24}$$

$$\frac{\partial L}{\partial \boldsymbol{\theta}_{k-3}} = \frac{\partial L}{\partial \mathbf{f}_k} \frac{\partial \mathbf{f}_k}{\partial \mathbf{f}_{k-1}} \boxed{\frac{\partial \mathbf{f}_{k-1}}{\partial \mathbf{f}_{k-2}} \frac{\partial \mathbf{f}_{k-2}}{\partial \boldsymbol{\theta}_{k-3}}} \tag{25}$$

$$\frac{\partial L}{\partial \boldsymbol{\theta}_{i}} = \frac{\partial L}{\partial \mathbf{f}_k} \frac{\partial \mathbf{f}_k}{\partial \mathbf{f}_{k-1}} \cdots \boxed{\frac{\partial \mathbf{f}_{i+2}}{\partial \mathbf{f}_{i+1}} \frac{\partial \mathbf{f}_{i+1}}{\partial \boldsymbol{\theta}_{i}}} \tag{26}$$

- Assuming, we have already computed the partial derivatives $\frac{\partial L}{\partial \boldsymbol{\theta}_{i+1}}$ then most of the computation can be reused to compute $\frac{\partial L}{\partial \boldsymbol{\theta}_i}$

Gradients in a Deep Network: Backpropagation



Source: M.P. Deisenroth *et al*, Mathematics for Machine Learning (First Edition)

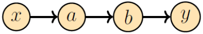
Automatic Differentiation

Automatic differentiation as a set of techniques to numerically evaluate the exact (up to machine precision) gradient of a function by working with intermediate variables and applying the chain rule.

- Automatic differentiation applies a series of elementary arithmetic operations, e.g., addition and multiplication and elementary functions e.g., sin; cos; exp; log.
- By applying the chain rule to these operations, the gradient of quite complicated functions can be computed automatically.
- There is a forward and reverse mode of automatic differentiation

Automatic Differentiation

Consider the simple graph representing the data flow from inputs x to outputs y via some intermediate variables a and b



Source: M.P. Deisenroth *et al*, Mathematics for Machine Learning (First Edition)

- Think of “a” and “b” as applying functions. If we were to compute the derivative $\mathrm{d}y/\mathrm{d}x$, we would apply the chain rule and obtain

$$\frac{\mathrm{d}y}{\mathrm{d}x} = \frac{\mathrm{d}y}{\mathrm{d}b}\frac{\mathrm{d}b}{\mathrm{d}a}\frac{\mathrm{d}a}{\mathrm{d}x} \tag{27}$$

- Given that we have associativity we can use the order or in one of two ways

$$\frac{\mathrm{d}y}{\mathrm{d}x} = \boxed{\frac{\mathrm{d}y}{\mathrm{d}b}\frac{\mathrm{d}b}{\mathrm{d}a}}\frac{\mathrm{d}a}{\mathrm{d}x} \qquad \text{Reverse mode} \tag{28}$$

$$= \frac{\mathrm{d}y}{\mathrm{d}b}\boxed{\frac{\mathrm{d}b}{\mathrm{d}a}\frac{\mathrm{d}a}{\mathrm{d}x}} \qquad \text{Forward mode} \tag{29}$$

Which to pick?

- In the context of neural networks, where the input dimensionality is often much higher than the dimensionality of the labels, **the reverse mode is computationally significantly cheaper** than the forward mode.

Automatic Differentiation: Worked example

Consider the function

$$f(x) = \sqrt{x^2 + e^{x^2}} + cos(x^2 + e^{x^2}) \tag{30}$$

If we were to implement a function f on a computer, we would be able to save some computation by using *intermediate variables*:

$$a = x^2, \tag{31}$$

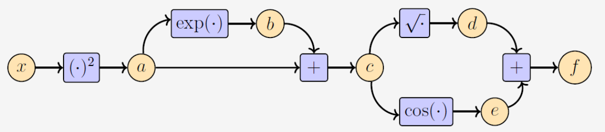
$$b = e^a, \tag{32}$$

$$c = a + b, \tag{33}$$

$$d = \sqrt{c}, \tag{34}$$

$$e = cos(c), \tag{35}$$

$$f = d + e. \tag{36}$$



Automatic Differentiation: Worked example

Now if we get all the derivatives of intermediate variables with respect to their "input"/independent variables we have

$$\frac{\partial a}{\partial x} = 2x \tag{37}$$

$$\frac{\partial b}{\partial a} = e^{a} \tag{38}$$

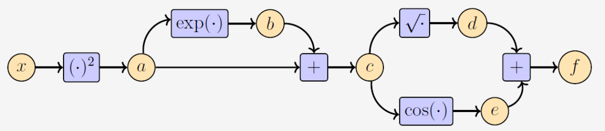
$$\frac{\partial c}{\partial a} = 1 = \frac{\partial c}{\partial b} \tag{39}$$

$$\frac{\partial d}{\partial c} = 0.5c^{-0.5} \tag{40}$$

$$\frac{\partial e}{\partial c} = -sin(c) \tag{41}$$

$$\frac{\partial f}{\partial d} = 1 = \frac{\partial f}{\partial e} \tag{42}$$

Automatic Differentiation: Worked example



We can now get the derivative of $\partial f/\partial x$ by working back through the graph, namely:

$$\frac{\partial f}{\partial c} = \frac{\partial f}{\partial d}\frac{\partial d}{\partial c} + \frac{\partial f}{\partial e}\frac{\partial e}{\partial c} \tag{43}$$

$$\frac{\partial f}{\partial b} = \frac{\partial f}{\partial c}\frac{\partial c}{\partial b} \tag{44}$$

$$\frac{\partial f}{\partial a} = \frac{\partial f}{\partial b}\frac{\partial b}{\partial a} + \frac{\partial f}{\partial c}\frac{\partial c}{\partial a} \tag{45}$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial a}\frac{\partial a}{\partial x} \tag{46}$$

Automatic Differentiation: Worked example

By substituting the results of the derivatives of the elementary functions, we get

$$\frac{\partial f}{\partial c} = 1 \cdot 0.5c^{-0.5} + 1 \cdot (-sin(c)) \tag{47}$$

$$\frac{\partial f}{\partial b} = \frac{\partial f}{\partial c} \cdot 1 \tag{48}$$

$$\frac{\partial f}{\partial a} = \frac{\partial f}{\partial b} e^{a} + \frac{\partial f}{\partial c} \cdot 1 \tag{49}$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial a} \cdot 2x \tag{50}$$

and therefore our desire derivative $\frac{\partial f}{\partial x}$

Higher-Order Derivatives

Second order derivatives, while computationally expensive are a fundamental tool in optimization (among many other). Some preliminary notation

- $\frac{\partial^2 f}{\partial x^2}$ is the second partial derivative of f with respect to x.
- $\frac{\partial^n f}{\partial x^n}$ is the nth partial derivative of f with respect to x.
- $\frac{\partial^2 f}{\partial y \partial x} = \frac{\partial}{\partial y}\frac{\partial f}{\partial x}$ is the partial derivative obtained by first partial differentiating with respect to x and then with respect to y.
- $\frac{\partial^2 f}{\partial x \partial y} = \frac{\partial}{\partial x}\frac{\partial f}{\partial y}$ is the partial derivative obtained by first partial differentiating by y and then x

Hessian

The **Hessian** is the collection of all second-order partial derivatives.

- if $f(x,y)$ is a twice (continuously) differentiable function then

$$\frac{\partial^2 f}{\partial x \partial y} = \frac{\partial^2 f}{\partial y \partial x} \tag{51}$$

  This means that the Hessian matrix if $f(x,y)$ is

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix} \tag{52}$$

  and is symmetric. The Hessian is denoted as $\nabla^2_{x,y} f(x,y)$
  - The Hessian measures the curvature of the function locally around (x,y).
  - If $f : \mathbb{R}^n \to \mathbb{R}^m$ is is a vector field, the Hessian is an $(m \times n \times n)$-tensor

Taylor Series

The Taylor series is **a** representation of a function f as an infinite sum of terms.

Taylor Polynomial

The *Taylor polynomial* of degree n of $f : \mathbb{R} \to \mathbb{R}$ at x_0 is defined as

$$T_n(x) := \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k \tag{12}$$

where $f^k(x_0)$ is the kth derivative of f at x_0 (under the assumption that the derivatives exist),

- $\frac{f^{(k)}(x_0)}{k!}$ are the coefficients of the polynomial.

Taylor Series

Taylor Series

For a smooth function $f \in \mathbf{C}^{\infty}$, $f : \mathbb{R} \to \mathbb{R}$, the Taylor series of f at x_0 is defined as

$$T_{\infty}(x) = \sum_{k=0}^{\infty} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k \tag{13}$$

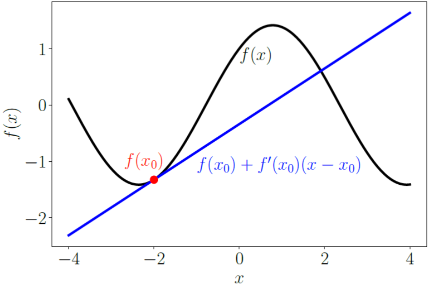
- For $x_0 = 0$, we obtain the *Maclaurin series* as a special instance of the Taylor series
- If $f(x) = T_{\infty}(x)$ then f is called analytic
- In general, a Taylor polynomial of degree n is an approximation of a function, which does not need to be a polynomial.
  - The Taylor polynomial is similar to f in a neighborhood around x_0.

The gradient ∇f of a function f is often used for a locally linear approximation of f around $\mathbf{x}_0$:

$$f(\mathbf{x}) \approx f(\mathbf{x}_0) + (\nabla_{\mathbf{x}} f)(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0) \tag{53}$$

Here$(\nabla_{\mathbf{x}} f)(\mathbf{x}_0)$ is the gradient of f with respect to $\mathbf{x}$, evaluated at $\mathbf{x}_0$.

- This approximation is locally accurate, but the farther we move away from $\mathbf{x}_0$ the worse the approximation gets.



Source: M.P. Deisenroth *et al.* Mathematics for Machine Learning (First Edition)

Multivariate Taylor Series

Consider,

$$f : \mathbb{R}^D \to \mathbb{R} \tag{54}$$

$$\mathbf{x} \mapsto f(\mathbf{x}), \mathbf{x} \in \mathbb{R}^D \tag{55}$$

that is smooth at x_0. When we define the difference vector $\boldsymbol{\delta} := \mathbf{x} - \mathbf{x}_0$, the multivariate Taylor series of f at $(\mathbf{x}_0)$ is defined as

$$f(\mathbf{x}) = \sum_{k=0}^{\infty} \frac{D_{\mathbf{x}}^k f(\mathbf{x_0})}{k!} \boldsymbol{\delta}^k \tag{56}$$

where $D_{\mathbf{x}}^k f(\mathbf{x}_0)$ is the k-th (total) derivative of f with respect to $\mathbf{x}$, evaluated at $\mathbf{x}_0$.

Taylor Polynomial

Taylor Polynomial

The Taylor polynomial of degree n of f at $\mathbf{x}_0$ contains the first n + 1 components of the series in equation (56) and is defined as

$$T_n(\mathbf{x}) = \sum_{k=0}^{n} \frac{D_{\mathbf{x}}^k f(\mathbf{x_0})}{k!} \boldsymbol{\delta}^k \tag{57}$$

Taylor Polynomial

The $\boldsymbol{\delta}^k$ term need to be properly defined.

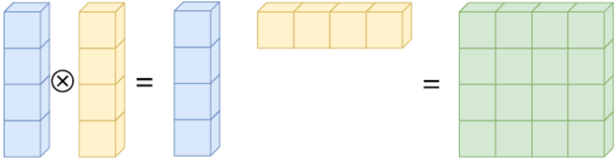
- Firstly, note that $D^k_{\mathbf{x}} f$ are $\boldsymbol{\delta}^k$ k-th order tensors.
- $\boldsymbol{\delta}^k \in \mathbb{R}^{\overbrace{D \times D \times \cdots \times D}^{\text{k times}}}$ is obtained as a k-fold outer product, denoted by $\otimes$, of the vector $\boldsymbol{\delta} \in \mathbb{R}^D$. For example,

$$\boldsymbol{\delta}^2 := \boldsymbol{\delta} \otimes \boldsymbol{\delta} = \boldsymbol{\delta}\boldsymbol{\delta}^T, \quad \boldsymbol{\delta}^2[i,j] = \delta[i]\delta[j] \tag{58}$$

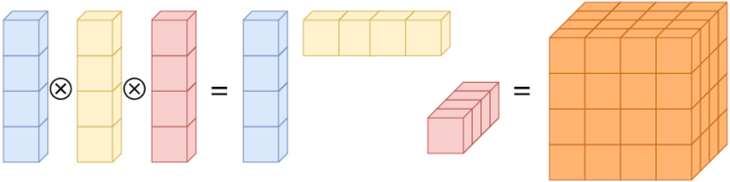
$$\boldsymbol{\delta}^3 := \boldsymbol{\delta} \otimes \boldsymbol{\delta} \otimes \boldsymbol{\delta}, \quad \boldsymbol{\delta}^3[i,j,k] = \delta[i]\delta[j]\delta[k] \tag{59}$$

$$\vdots$$

$$\boldsymbol{\delta}^n := \overbrace{\boldsymbol{\delta} \otimes \cdots \otimes \boldsymbol{\delta}}^{\text{n times}}, \quad \boldsymbol{\delta}^n[i_1, i_2, \ldots, i_n] = \delta[i_1]\delta[i_2]\cdots\delta[i_n] \tag{60}$$



(a) Given a vector $\delta \in \mathbb{R}^4$, we obtain the outer product $\delta^2 := \delta \otimes \delta = \delta\delta^\top \in \mathbb{R}^{4\times 4}$ as a matrix.



(b) An outer product $\delta^3 := \delta \otimes \delta \otimes \delta \in \mathbb{R}^{4\times 4\times 4}$ results in a third-order tensor ("three-dimensional matrix"), i.e., an array with three indexes.

Source: M.P. Deisenroth *et al*, Mathematics for Machine Learning (First Edition)

Taylor Polynomial

In general, we obtain the terms

$$D_{\mathbf{x}}^k f(\mathbf{x}_0)\boldsymbol{\delta}^k = \sum_{i_1=1}^{D} \cdots \sum_{i_k=1}^{D} D_{\mathbf{x}}^k f(\mathbf{x}_0)[i_1, \ldots, i_k]\delta[i_1] \cdots \delta[i_k] \tag{61}$$

in the Taylor series, where $D_{\mathbf{x}}^k f(\mathbf{x}_0)\boldsymbol{\delta}^k$ contains k-th order polynomials.

Taylor Polynomial

Now that we defined the Taylor series for vector fields, let us explicitly write down the first terms $D_{\mathbf{x}}^k f(\mathbf{x}_0)\boldsymbol{\delta}^k$ of the Taylor series expansion for $k = 0, \ldots, 3$ and $\boldsymbol{\delta} := \mathbf{x} - \mathbf{x}_0$

$$k = 0 : D_{\boldsymbol{x}}^0 f(\boldsymbol{x}_0)\boldsymbol{\delta}^0 = f(\boldsymbol{x}_0) \in \mathbb{R}$$

$$k = 1 : D_{\boldsymbol{x}}^1 f(\boldsymbol{x}_0)\boldsymbol{\delta}^1 = \underbrace{\nabla_{\boldsymbol{x}} f(\boldsymbol{x}_0)}_{1\times D} \underbrace{\boldsymbol{\delta}}_{D\times 1} = \sum_{i=1}^{D} \nabla_{\boldsymbol{x}} f(\boldsymbol{x}_0)[i]\delta[i] \in \mathbb{R}$$

$$k = 2 : D_{\boldsymbol{x}}^2 f(\boldsymbol{x}_0)\boldsymbol{\delta}^2 = \operatorname{tr}\big(\underbrace{\boldsymbol{H}(\boldsymbol{x}_0)}_{D\times D} \underbrace{\boldsymbol{\delta}}_{D\times 1} \underbrace{\boldsymbol{\delta}^\top}_{1\times D}\big) = \boldsymbol{\delta}^\top \boldsymbol{H}(\boldsymbol{x}_0)\boldsymbol{\delta}$$

$$= \sum_{i=1}^{D} \sum_{j=1}^{D} H[i,j]\delta[i]\delta[j] \in \mathbb{R}$$

$$k = 3 : D_{\boldsymbol{x}}^3 f(\boldsymbol{x}_0)\boldsymbol{\delta}^3 = \sum_{i=1}^{D} \sum_{j=1}^{D} \sum_{k=1}^{D} D_x^3 f(\boldsymbol{x}_0)[i,j,k]\delta[i]\delta[j]\delta[k] \in \mathbb{R}$$

Homework: Work through 5.15.