Lecture #18: Automatic Differentiation

AM 207: Advanced Scientific Computing

Stochastic Methods for Data Analysis, Inference and Optimization

Fall, 2020

IACS

VE RI TAS

Outline

1. Review of BBVI
2. Automatic Differentiation

Review of Black Box Variational Inference

Developments in Computationally Efficient Variational Inference

1. **Black-box Variational Inference (2013)** Uses the log-derivative trick to rewrite the gradient of the ELBO as:

$$\nabla_{\mu,\Sigma}\, ELBO(\mathbf{W})$$
$$= \mathbb{E}_{\mathbf{W}\sim q(\mathbf{W}|\mu,\Sigma)}\left[\nabla_{\mu,\Sigma}\, q(\mathbf{W}|\mu,\Sigma) * \log\left(\frac{p(\mathbf{W})\prod_{n=1}^{N} p(Y^{(n)}|\mathbf{X}^{(n)})}{q(\mathbf{W}|\mu,\Sigma)}\right)\right]$$

This requires **only** the computation of the gradient of $q(\mathbf{W})$, which is generally much simpler than $p(\mathbf{W})\prod_{n=1}^{N} p(Y^{(n)}|\mathbf{X}^{(n)})$.

Implementation of BBVI means hard-coding a large library of different kinds of variational distributions $q(\mathbf{W}|\lambda)$ and their gradients. User inputs the joint distribution of their Bayesian model and chooses a variational family $q(\mathbf{W}|\lambda)$ - then you can optimize the variational parameters λ to best approximate the target posterior by gradient descent.

Developments in Computationally Efficient Variational Inference

Weight Uncertainty in Neural Networks (2015) Assuming the variational family is mean-field Gaussian, uses the reparametrization trick to rewrite the gradient of the ELBO as:

$$\nabla_{\mu,\Sigma} \, ELBO(\mathbf{W}) = \mathbb{E}_{\epsilon \sim \mathcal{N}(0,\mathbf{I})} \left[\nabla_{\mu,\Sigma} \log \left[p(\epsilon^\top \Sigma^{1/2} + \mu) \prod_{n=1}^{N} p(Y^{(n)} | \mathbf{X}^{(n)}, \epsilon^\top \Sigma^{1/2} + \mu) \right] \right] + \nabla_{\mu,\Sigma} \underbrace{\mathbb{E}_{\mathbf{W} \sim \mathcal{N}(\mu,\Sigma)} \left[\log \mathcal{N}(\mathbf{W}; \mu, \Sigma) \right]}_{\text{Guassian entropy: has closed form}}$$

For Bayesian Neural Networks, the gradient in the above can be computed by backpropagation - then you can optimize the variational parameters μ, Σ to best approximate the target posterior by gradient descent. This algorithm is called **Bayes by Backprop**.

Developments in Computationally Efficient Variational Inference

1. **Automatic Differentiation Variational Inference (2016)** Anytime the gradient of the ELBO can be written as the expectation of a gradient (of an expression without integrals), the gradient can be computed by any automatic differentiation package - then you can optimize the variational parameters λ to best approximate the target posterior by gradient descent.

Automatic Differentiation

Types of Computational Differentiation

1. Manually computing closed form expressions of derivatives and then coding them up (say using `numpy` functions).

2. **numeric differentiation:** approximate a derivative $\frac{df(x)}{dx}$ with a rate of change $\frac{f(x+h)-f(x)}{h}$ at $x = a$.

3. **symbolic differentiation:** manipulate expressions of functions using pre-programmed rules (of differentiation).

4. **automatic differentiation:** algorithmic computation of exact numeric derivatives. `autograd` is a `python` implementation of automatic differentiation. `pytorch` implements one particular mode of automatic differentiation, also called `autograd`.

Numeric Differentiation

Given $f : \mathbb{R}^n \to \mathbb{R}^m$, approximate each gradient $\nabla f_j = \left(\frac{\partial f_j}{\partial x_i} \dots \frac{\partial f_j}{\partial x_n} \right)$ for $i = 1, \dots, n$ and $j = 1, \dots, m$ with

$$\frac{\partial f_j}{\partial x_i} \approx \frac{f_j(x + he_i) - f_j(x)}{h}$$

where e_i is the i-th standard basis vector of $\mathbb{R}^n$.

This is numerically unstable when $h \approx 0$ and biased when h is large. For each gradient ∇f_j, it requires $O(n)$ computations.

Symbolic Differentiation

Given an expression for a function $f : \mathbb{R}^n \to \mathbb{R}^m$, we represent the expression as a tree and automatically manipulate the expression tree by applying transformations representing of differentiation:

$$\frac{d}{dx}[h(x) + g(x)] \to \frac{d}{dx}h(x) + \frac{d}{dx}g(x)$$

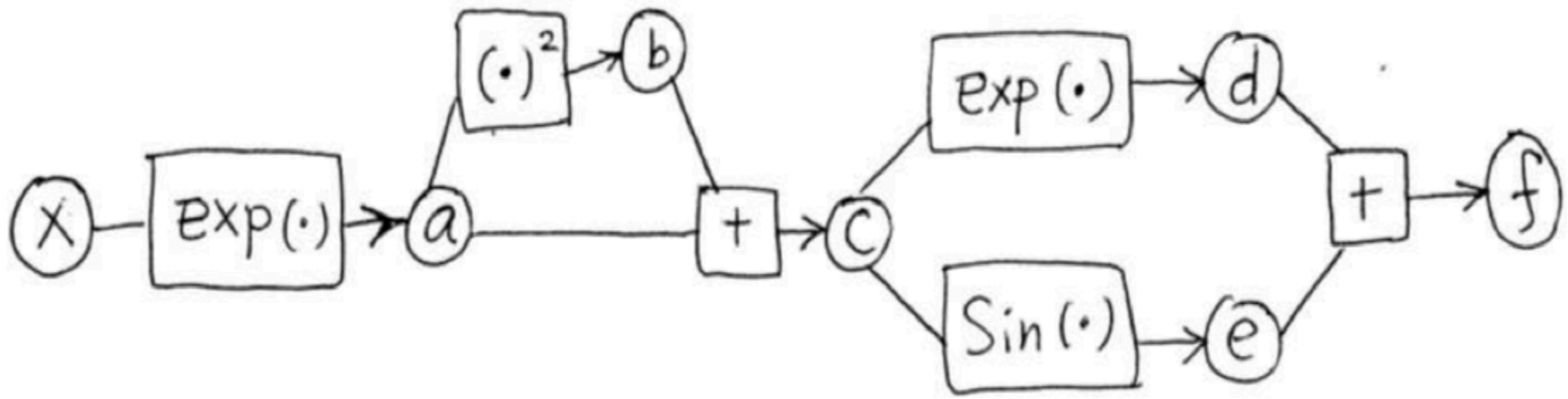
This can be computationally inefficient since expressions of derivatives can be exponentially longer than the original function expression:

$$f(x) = h(x)^2 g(x) + \ln(h(x)) + g(x)$$
$$f'(x) = 2h(x)h'(x)g(x) + h(x)^2 g'(x) + \frac{h'(x)}{h(x)} + g'(x)$$

Numerical evaluation of the derivative can be inefficient due to the redudant evaluation of the components of f.

Automatic Differentiation: The Idea

We decompose a function $f(a, b) = \sin(e^a + e^{2a}) + e^{(e^a + e^{2a})}$ into elementary operations:



We apply symbolic differentiation to elementary operations, like: arithmetic operations, elementary functions (exponential, logarithmic, trignometric, power).

We keep intermediate values of the components of f so that they can be reused.

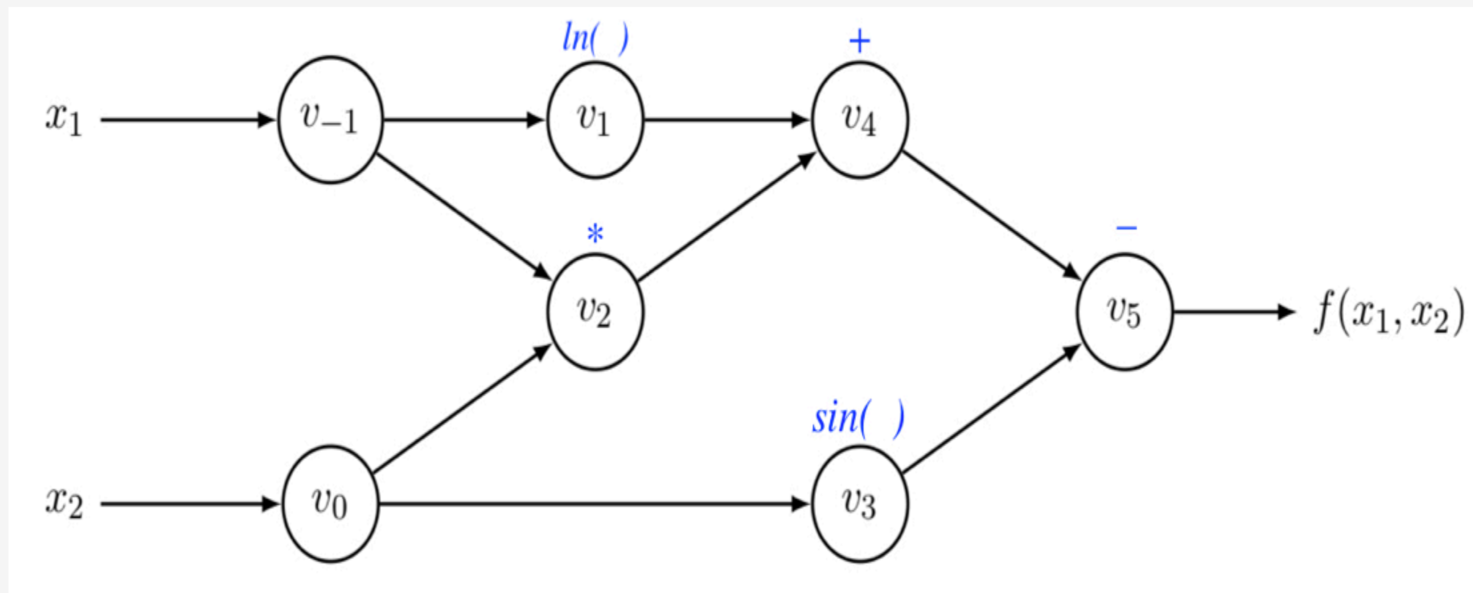
Evaluation Trace and Computational Graph

Given a function $f : \mathbb{R}^n \to \mathbb{R}^m$, we represent f as the composition of elementary functions through elemtary operations by a sequence of intermediate values v_k that is involved with the evaluation of f, this is the ***evaluation trace***. We can also represent the trace graphically, resulting in the ***computational graph***.

Example: Given $y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - \sin(x_2)$, its evaluation trace and computational graph are:

Forward

v_{-1}	$= x_1$	$= 2$
v_0	$= x_2$	$= 5$
v_1	$= \ln v_{-1}$	$= \ln 2$
v_2	$= v_{-1} \times v_0$	$= 2 \times 5$
v_3	$= \sin v_0$	$= \sin 5$
v_4	$= v_1 + v_2$	$= 0.693 + 10$
v_5	$= v_4 - v_3$	$= 10.693 + 0.959$
y	$= v_5$	$= 11.652$

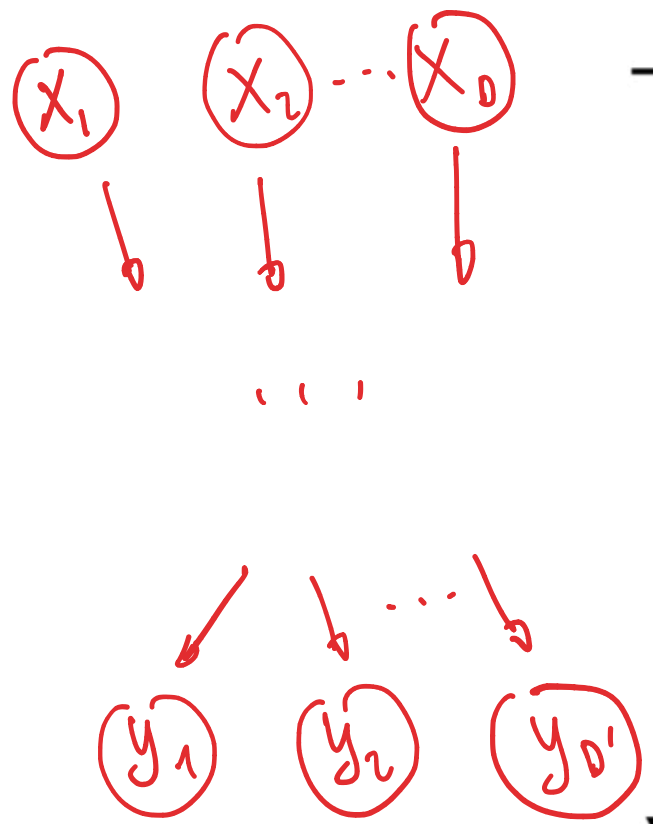


Automatic Differentiation: Forward Mode

In ***forward mode automatic differentiation***, we start with in the input and work towards the output: evaluating the value of each intermediate value v_k as well as the derivative of v_k with repect to a fixed x_i using the chain rule:

$$\frac{\partial v_k}{\partial x_i} = \sum_{v \in \text{parent}(v_k)} \frac{\partial v_k}{\partial v} \frac{\partial v}{\partial x_i}$$

We denote $\frac{\partial v_k}{x_i}$ by $\dot{v}_k$.



Forward		
v_{-1}	$= x_1$	$= 2$
v_0	$= x_2$	$= 5$
v_1	$= \ln v_{-1}$	$= \ln 2$
v_2	$= v_{-1} \times v_0$	$= 2 \times 5$
v_3	$= \sin v_0$	$= \sin 5$
v_4	$= v_1 + v_2$	$= 0.693 + 10$
v_5	$= v_4 - v_3$	$= 10.693 + 0.959$
y	$= v_5$	$= 11.652$

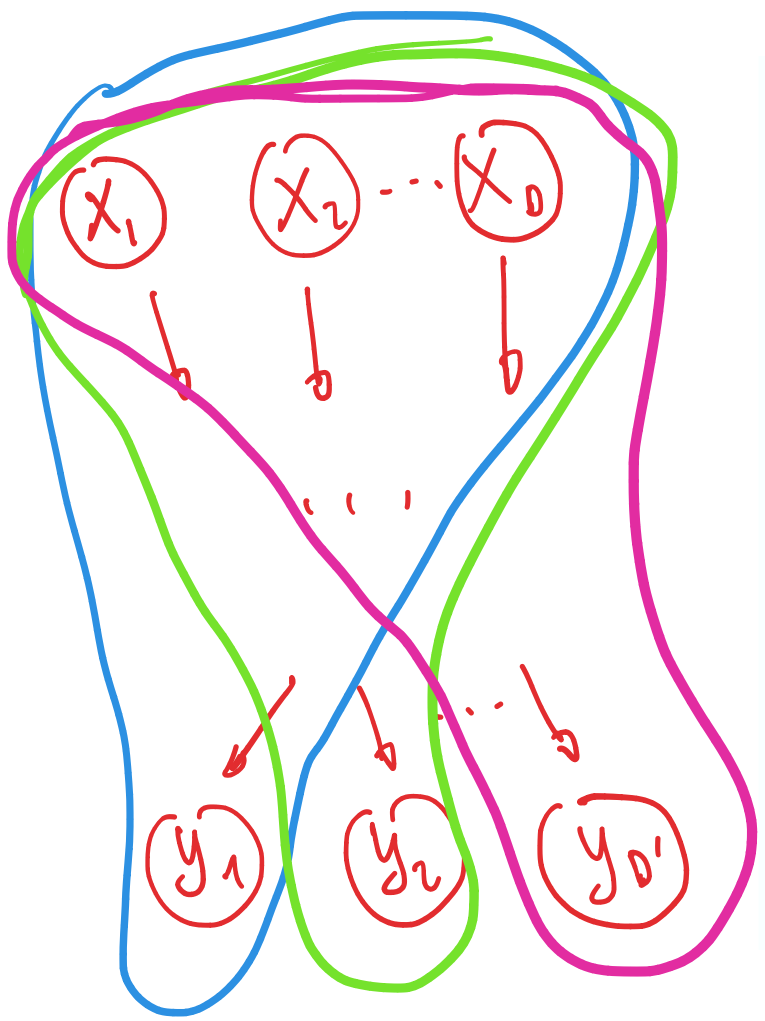
Forward Tangent (Derivative)			
$\dot{v}_{-1}$	$= \dot{x}_1$	$= 1$	$\frac{\partial f}{\partial x_1}$
$\dot{v}_0$	$= \dot{x}_2$	$= 0$	$\frac{\partial f}{\partial x_2}$
$\dot{v}_1$	$= \dot{v}_{-1} / v_{-1}$	$= 1/2$	
$\dot{v}_2$	$= \dot{v}_{-1} \times v_0 + \dot{v}_0 \times v_{-1}$	$= 1 \times 5 + 0 \times 2$	
$\dot{v}_3$	$= \dot{v}_0 \times \cos v_0$	$= 0 \times \cos 5$	
$\dot{v}_4$	$= \dot{v}_1 + \dot{v}_2$	$= 0.5 + 5$	
$\dot{v}_5$	$= \dot{v}_4 - \dot{v}_3$	$= 5.5 - 0$	
$\boldsymbol{\dot{y}}$	$\boldsymbol{= \dot{v}_5}$	$\mathbf{= 5.5}$	

Automatic Differentiation: Reverse Mode

In ***reverse mode automatic differentiation***, we first do a forward pass to compute all intermediate values. Then we start with in the output and work towards the input: evaluating the derivative of f with repect to an intermediate value v_k using the chain rule:

$$\frac{\partial f}{\partial v_k} = \sum_{v \in \text{child}(v_k)} \frac{\partial f}{\partial v} \frac{\partial v}{\partial v_k}$$

We denote $\frac{\partial f}{v_k}$ by $\overline{v}_k$.



Forward		
v_{-1}	$= x_1$	$= 2$
v_0	$= x_2$	$= 5$
v_1	$= \ln v_{-1}$	$= \ln 2$
v_2	$= v_{-1} \times v_0$	$= 2 \times 5$
v_3	$= \sin v_0$	$= \sin 5$
v_4	$= v_1 + v_2$	$= 0.693 + 10$
v_5	$= v_4 - v_3$	$= 10.693 + 0.959$
y	$= v_5$	$= 11.652$

Reverse (Derivative)			
$\boldsymbol{\bar{x}_1}$	$\boldsymbol{= \bar{v}_{-1}}$		$\mathbf{= 5.5}$
$\boldsymbol{\bar{x}_2}$	$\boldsymbol{= \bar{v}_0}$		$\mathbf{= 1.716}$
$\bar{v}_{-1}$	$= \bar{v}_{-1} + \bar{v}_1 \frac{\partial v_1}{\partial v_{-1}}$	$= \bar{v}_{-1} + \bar{v}_1 / v_{-1}$	$= 5.5$
$\bar{v}_0$	$= \bar{v}_0 + \bar{v}_2 \frac{\partial v_2}{\partial v_0}$	$= \bar{v}_0 + \bar{v}_2 \times v_{-1}$	$= 1.716$
$\bar{v}_{-1}$	$= \bar{v}_2 \frac{\partial v_2}{\partial v_{-1}}$	$= \bar{v}_2 \times v_0$	$= 5$
$\bar{v}_0$	$= \bar{v}_3 \frac{\partial v_3}{\partial v_0}$	$= \bar{v}_3 \times \cos v_0$	$= -0.284$
$\bar{v}_2$	$= \bar{v}_4 \frac{\partial v_4}{\partial v_2}$	$= \bar{v}_4 \times 1$	$= 1$
$\bar{v}_1$	$= \bar{v}_4 \frac{\partial v_4}{\partial v_1}$	$= \bar{v}_4 \times 1$	$= 1$
$\bar{v}_3$	$= \bar{v}_5 \frac{\partial v_5}{\partial v_3}$	$= \bar{v}_5 \times (-1)$	$= -1$
$\frac{\partial y}{\partial v_4}$ $\bar{v}_4$	$= \bar{v}_5 \frac{\partial v_5}{\partial v_4}$	$= \bar{v}_5 \times 1$	$= 1$
$\bar{v}_5$	$= \bar{y}$ $\frac{\partial y}{\partial v_5}$	$= 1$	

Processing math: 100%

Implementing Reverse Mode AutoDiff

We see that each intermediate gradient computation $\frac{\partial f}{\partial v_k}$ in reverse mode autodiff is local, it only requires:

1. the current value of v_k
2. the derivative of f with respect to every child of v_k: $\frac{\partial f}{\partial v},\ v \in \mathrm{child}(v_k)$.
3. the derivative of the elementary function h_v describing the way v depends on v_k.

We implement the computation graph of a function f as a directed graph, where each node keeps tracks of the above three pieces of information and uses them to compute its own gradient.