BDMI-课程编号-01510243

自动微分
Automatic Differentiation

Shiyu Huang

huangsy1314@163.com

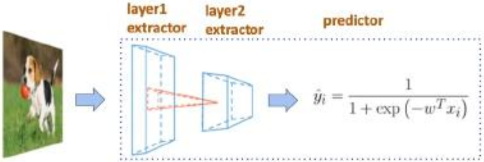
https://huangshiyu13.github.io

Dept. of Comp. Sci. Tech., BNRist Center, Institute for AI, THBI Lab, Tsinghua University

Table Of Contents

Outline

Background



Objective

$$L(\omega) = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \lambda \parallel \omega \parallel^2$$

Training

$$\omega \leftarrow \omega - \eta \nabla_\omega L(\omega)$$

Differentiation

To get the gradient, we have the following differentiation methods:

- Manual Differentiation

Differentiation

To get the gradient, we have the following differentiation methods:

- Manual Differentiation
- Numerical Differentiation

Differentiation

To get the gradient, we have the following differentiation methods:

- Manual Differentiation
- Numerical Differentiation
- Symbolic Differentiation

Differentiation

To get the gradient, we have the following differentiation methods:

- Manual Differentiation
- Numerical Differentiation
- Symbolic Differentiation
- Automatic Differentiation

Manual Differentiation

Manually work out derivatives and code them. For example, in an old MATLAB optimization program, sometimes you have to provide a “gradient function” aside with the loss function.

```
function [x,fval]=opfgrad
x0=[-1,1];%初始值如何设置
A=[];b=[];%Ax<b
Aeq=[];beq=[];%Aeq.x=beq
lb=[];ub=[];%搜索范围
options=optimset('Largescale','off');
options=optimset(options,'gradobj','on','gradconstr','on');
%LargeScale 指大规模搜索. off 表示在规模搜索模式关闭. Simplex 指单纯形算法.
on 表示该算法打开
[x,fval]=fmincon(@fobj,x0,A,b,Aeq,beq,lb,ub,@confgrad,options);

function [f,g]=fobj(x)%目标函数以及相应的梯度
f=exp(x(1))*(4*x(1)^2+2*x(2)^2+4*x(1)*x(2)+2*x(2)+1);%目标函数
fx1=f+exp(x(1))*(8*x(1)+4*x(2));%目标函数的梯度 1
fx2=exp(x(1))*(4*x(1)+4*x(2)+2);%目标函数的梯度 2
g=[fx1;fx2];%目标函数的梯度

function [c,ceq,gcon,gceq]=confgrad(x)
fcon1=1.5+x(1)*x(2)-x(1)-x(2);%约束函数 1
gxcon1=[x(2)-1,-x(2)];%约束函数 1 梯度
fcon2=-x(1)*x(2)-10;%约束函数 2
gxcon2=[x(1)-1,-x(1)];%约束函数 2 梯度
c=[fcon1;fcon2];%约束函数
```

Numerical Differentiation

Use finite difference approximations.

$$\frac{\partial f(\mathbf{x})}{\partial x_i} \approx \frac{f(\mathbf{x} + h\mathbf{e}_i) - f(\mathbf{x})}{h}, \quad h \to 0$$

$$\frac{\partial f(\mathbf{x})}{\partial x_i} \approx \frac{f(\mathbf{x} + h\mathbf{e}_i) - f(\mathbf{x} - h\mathbf{e}_i)}{2h}, \quad h \to 0$$

Numerical Differentiation

Use finite difference approximations.

$$\frac{\partial f(\mathbf{x})}{\partial x_i} \approx \frac{f(\mathbf{x}+h\mathbf{e}_i) - f(\mathbf{x})}{h}, \quad h \to 0$$

$$\frac{\partial f(\mathbf{x})}{\partial x_i} \approx \frac{f(\mathbf{x}+h\mathbf{e}_i) - f(\mathbf{x}-h\mathbf{e}_i)}{2h}, \quad h \to 0$$

Simple to implement, but

- highly inaccurate due to round-off and truncation errors
- scales poorly for gradients

Symbolic Differentiation

Use expression manipulation in computer algebra systems such as Mathematica, Maxima and Maple.

$$\frac{d}{dx}(f(x) + g(x)) = \frac{d}{dx}f(x) + \frac{d}{dx}g(x)$$

$$\frac{d}{dx}(f(x)g(x)) = (\frac{d}{dx}f(x))g(x) + f(x)(\frac{d}{dx}g(x))$$

Symbolic Differentiation

Use expression manipulation in computer algebra systems such as Mathematica, Maxima and Maple.

$$\frac{d}{dx}(f(x)+g(x)) = \frac{d}{dx}f(x) + \frac{d}{dx}g(x)$$

$$\frac{d}{dx}(f(x)g(x)) = (\frac{d}{dx}f(x))g(x) + f(x)(\frac{d}{dx}g(x))$$

Address the weaknesses above, but

- "expression swell" for complex and cryptic expressions

n	l_n	$\frac{d}{dx}l_n$	$\frac{d}{dx}l_n$ (Simplified form)
1	x	1	1
2	$4x(1-x)$	$4(1-x)-4x$	$4-8x$
3	$16x(1-x)(1-2x)^2$	$16(1-x)(1-2x)^2-16x(1-2x)^2-64x(1-x)(1-2x)$	$16(1-10x+24x^2-16x^3)$
4	$64x(1-x)(1-2x)^2(1-8x+8x^2)^2$	$128x(1-x)(-8+16x)(1-2x)^2(1-8x+8x^2)+64(1-x)(1-2x)^2(1-8x+8x^2)^2-64x(1-2x)^2(1-8x+8x^2)^2-256x(1-x)(1-2x)(1-8x+8x^2)^2$	$64(1-42x+504x^2-2640x^3+7040x^4-9984x^5+7168x^6-2048x^7)$

Symbolic Differentiation

Use expression manipulation in computer algebra systems such as Mathematica, Maxima and Maple.

$$\frac{d}{dx}(f(x)+g(x)) = \frac{d}{dx}f(x) + \frac{d}{dx}g(x)$$

$$\frac{d}{dx}(f(x)g(x)) = (\frac{d}{dx}f(x))g(x) + f(x)(\frac{d}{dx}g(x))$$

Address the weaknesses above, but

- "expression swell" for complex and cryptic expressions

n	l_n	$\frac{d}{dx}l_n$	$\frac{d}{dx}l_n$ (Simplified form)
1	x	1	1
2	$4x(1-x)$	$4(1-x)-4x$	$4-8x$
3	$16x(1-x)(1-2x)^2$	$16(1-x)(1-2x)^2-16x(1-2x)^2-64x(1-x)(1-2x)$	$16(1-10x+24x^2-16x^3)$
4	$64x(1-x)(1-2x)^2(1-8x+8x^2)^2$	$128x(1-x)(-8+16x)(1-2x)^2(1-8x+8x^2)+64(1-x)(1-2x)^2(1-8x+8x^2)^2-64x(1-2x)^2(1-8x+8x^2)^2-256x(1-x)(1-2x)(1-8x+8x^2)^2$	$64(1-42x+504x^2-2640x^3+7040x^4-9984x^5+7168x^6-2048x^7)$

- models have to be defined as closed–form expressions

Automatic Differentiation

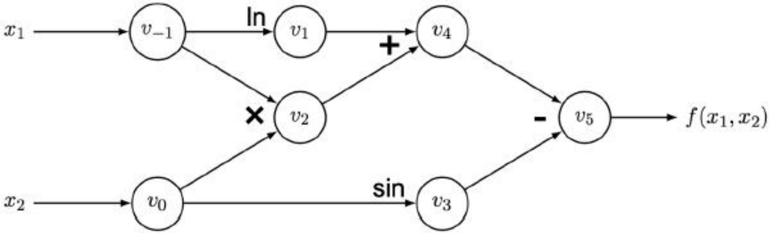
Apply chain rule to the **computation graph** (CG), which is a directed acyclic graph (**DAG**).

$$f(x_1, x_2) = \ln(x_1) + x_1 x_2 - \sin(x_2)$$

Automatic Differentiation

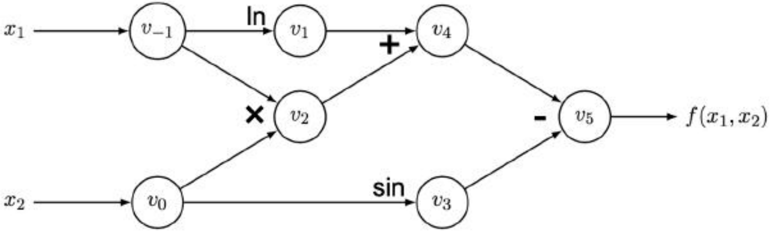
Apply chain rule to the **computation graph** (CG), which is a directed acyclic graph (**DAG**).

$$f(x_1, x_2) = \ln(x_1) + x_1 x_2 - \sin(x_2)$$



Outline

Forward Mode

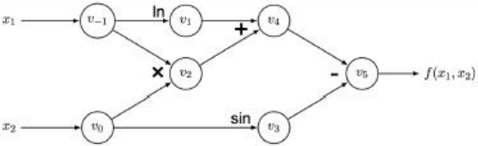


Suppose we want to get $\partial y_1 / \partial x_1$, just set $\dot{\mathbf{x}} = \mathbf{e}_1$ and traverse CG(DAG) in order:

$$\dot{v}_i = \frac{\partial v_i}{\partial x_1} = \sum_{j \in IE(i)} \frac{\partial v_i}{\partial v_j} \dot{v}_j$$

with $IE(i)$ is the collection of adjoints from input edges of v_i.

Forward Mode



Forward Primal Trace		
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) Trace		
$\dot{v}_{-1}$	$= \dot{x}_1$	$= 1$
$\dot{v}_0$	$= \dot{x}_2$	$= 0$
$\dot{v}_1$	$= \dot{v}_{-1} / v_{-1}$	$= 1/2$
$\dot{v}_2$	$= \dot{v}_{-1} \times v_0 + \dot{v}_0 \times v_{-1}$	$= 1 \times 5 + 0 \times 2$
$\dot{v}_3$	$= \dot{v}_0 \times \cos v_0$	$= 0 \times \cos 5$
$\dot{v}_4$	$= \dot{v}_1 + \dot{v}_2$	$= 0.5 + 5$
$\dot{v}_5$	$= \dot{v}_4 - \dot{v}_3$	$= 5.5 - 0$
$\boldsymbol{\dot{y}}$	$\boldsymbol{= \dot{v}_5}$	$\mathbf{= 5.5}$

Forward Mode

If we want:

$$\mathbf{J}_f \cdot \mathbf{r} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix} \begin{bmatrix} r_1 \\ \vdots \\ r_n \end{bmatrix}$$

Forward Mode

If we want:

$$\mathbf{J}_f \cdot \mathbf{r} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix} \begin{bmatrix} r_1 \\ \vdots \\ r_n \end{bmatrix}$$

Just feed $\mathbf{r}$ to the input nodes:

$$\dot{\mathbf{x}} = \mathbf{r}$$

and traverse CG once.

Forward Mode

Q: What if we want $\partial \mathbf{y} / \partial x_1$?

Forward Mode

Q: What if we want $\partial \mathbf{y} / \partial x_1$?

A: We already have $\partial \mathbf{y} / \partial x_1$.

Forward Mode

Q: What if we want $\partial \mathbf{y} / \partial x_1$?
A: We already have $\partial \mathbf{y} / \partial x_1$.

Q: What if we want $\partial y_1 / \partial \mathbf{x}$?

Forward Mode

Q: What if we want $\partial \mathbf{y} / \partial x_1$?
A: We already have $\partial \mathbf{y} / \partial x_1$.

Q: What if we want $\partial y_1 / \partial \mathbf{x}$?
A: You have to get $\partial y_1 / \partial x_i$ one by one.

Outline

Forward Mode – Dual Value

Mathematically, forward mode AD can be viewed as evaluating a function using dual numbers, which can be defined as truncated Taylor series of the form

$$v + \dot{v}\epsilon$$

where $v, \dot{v} \in \mathbb{R}$and ϵ is a nilpotent number such that $\epsilon^2 = 0$ and $\epsilon \neq 0$.
In the program, we can use a user–defined class to replace the original variable, with operators defined as follows.

Forward Mode – Dual Value

We can utilize this by setting up a regime where

$$f(v + i\epsilon) = f(v) + f'(v)i\epsilon$$

Also we have the chain rule as

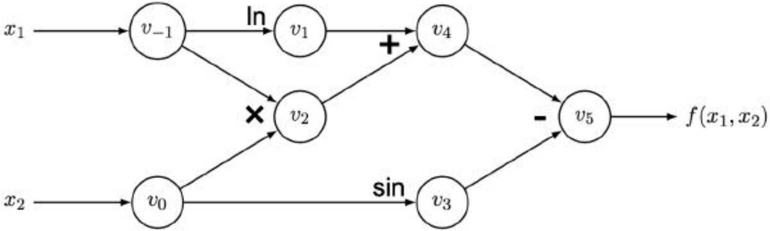
$$f(g(v + i\epsilon)) = f(g(v) + g'(v)i\epsilon) = f(g(v)) + f'(g(v))g'(v)i\epsilon.$$

We can extract the derivative of a function by interpreting any non–dual number v as $v + 0\epsilon$ and evaluating the function in this non–standard way on an initial input with a coefficient 1 for ϵ:

$$\left.\frac{df(x)}{dx}\right|_{x=v} = \text{epsilon–coefficient(dual–version}(f)(v + 1\epsilon))$$

Outline

Reverse Mode – Backpropagation

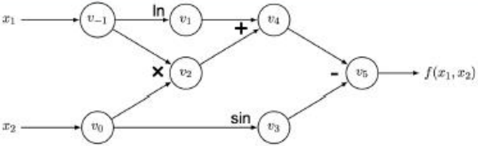


Suppose we want to get $\partial y_1/\partial x_1$, just set $\bar{\mathbf{y}} = \mathbf{e}_1$ and traverse CG(DAG) in reverse order:

$$\bar{v}_i = \frac{\partial y_1}{\partial v_i} = \sum_{j \in OE(i)} \frac{\partial v_j}{\partial v_i} \bar{v}_j$$

with $OE(i)$ is the collection of adjoints from output edges of v_i.

Reverse Mode – Backpropagation



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

Reverse Mode – Backpropagation

If we want:

$$\mathbf{J}_f^T \cdot \mathbf{r} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_1}{\partial x_n} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix} \begin{bmatrix} r_1 \\ \vdots \\ r_n \end{bmatrix}$$

Just feed $\mathbf{r}$ to the reverse input nodes:

$$\bar{\mathbf{y}} = \mathbf{r}$$

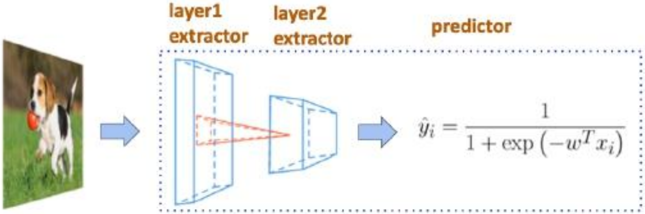
and traverse CG once.

Reverse Mode – Backpropagation

We already have $\partial y_1/\partial \mathbf{x}$ but have to calculate $\partial \mathbf{y}/\partial x_1$ one by one.

Reverse Mode – Backpropagation

We already have $\partial y_1/\partial \mathbf{x}$ but have to calculate $\partial \mathbf{y}/\partial x_1$ one by one.
Considering the real world ......



Reverse mode is preferred!

Outline

Reverse Mode – Graph

Is it enough?

Can we do better regrading the algorithm?

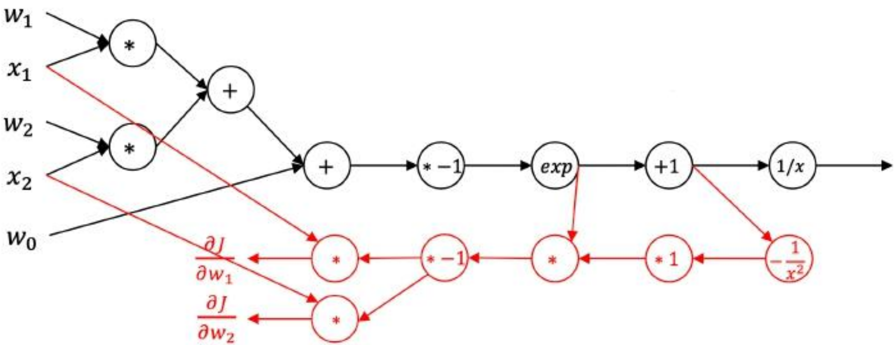
Problems of backpropagation

- You always need to keep intermediate data in the memory during the forward pass in case it will be used in the backpropagation.
- Lack of flexibility, e.g., compute the gradient of gradient.

Reverse Mode – Graph

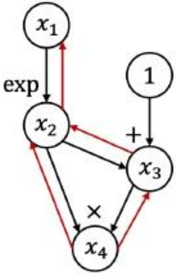
Create computation graph for gradient computation

$$f_w(x) = \frac{1}{1 + \exp\left(-\left(w_0 + w_1 x_1 + w_2 x_2\right)\right)}$$

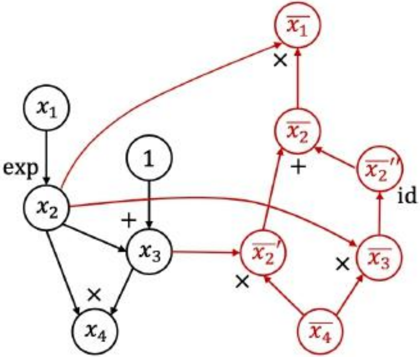


Reverse Mode – Graph

Backpropagation



AutoDiff



References I

- CSE599W: Spring 2018. Lecture 4: Backpropagation and Automatic Differentiation. 2018.
- Atılım Günes Baydin et al. "Automatic differentiation in machine learning: a survey". In: The Journal of Machine Learning Research 18.1 (2017), pp. 5595—5637.
- hunkim. PyTorchZeroToAll. 2019. URL: https://github.com/hunkim/PyTorchZeroToAll (visited on 04/01/2020).

Thanks~

Questions?