Calculating Gauss-Newton Matrix-Vector Product by Vector-Jacobian Products

Pengrui Quan
UCLA

November 5, 2019

1 Introduction

In a Newton method for training deep neural networks, at each conjugate gradient step a Gauss-Newton matrix-vector product is conducted. Here we discuss our implementation by using two vector-Jacobian products, a type of operations that are commonly available in packages like Tensorflow. Our derivation follows from the description at `https://j-towns.github.io/2017/06/12/A-new-trick.html` [2], though we follow the notation in Wang et al [3].

2 Deriving right multiplication of a Jacobian matrix

The key procedure of calculating $G\boldsymbol{v}$ is to derive the right multiplication of Jacobian matrices. In most deep learning libraries, the product of left multiplication of a Jacobian matrix, denoted as $\boldsymbol{v}^T J_\theta$, is well established, i.e. Tensorflow, PyTorch. We can make use of techniques explained below to develop the right multiplication of the Jacobian matrix $J_\theta \boldsymbol{v}$.

First we denote $f(\boldsymbol{\theta}) : \mathbb{R}^n \to \mathbb{R}^m$ with parameters $\boldsymbol{\theta} \in \mathbb{R}^n$. Let $\boldsymbol{v} \in \mathbb{R}^n$ be the right multiplication vector and $\boldsymbol{u} \in \mathbb{R}^m$ be a dummy variable.

The Jacobian matrix of f with respect to $\boldsymbol{\theta}$ is:

$$J_\theta = \begin{bmatrix} \frac{\partial f_1}{\partial \theta_1} & \frac{\partial f_1}{\partial \theta_2} & \frac{\partial f_1}{\partial \theta_3} & \cdots & \frac{\partial f_1}{\partial \theta_n} \\ \frac{\partial f_2}{\partial \theta_1} & \frac{\partial f_2}{\partial \theta_2} & \frac{\partial f_2}{\partial \theta_3} & \cdots & \frac{\partial f_2}{\partial \theta_n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial \theta_1} & \frac{\partial f_m}{\partial \theta_2} & \frac{\partial f_m}{\partial \theta_3} & \cdots & \frac{\partial f_m}{\partial \theta_n} \end{bmatrix} \tag{1}$$

The left multiplication of the Jacobian matrix can be calculated by back propagation, which is common in reverse mode automatic differentiation packages [1, p. 12]:

$$\boldsymbol{u}^T \frac{\partial f}{\partial \boldsymbol{\theta}^T} = \boldsymbol{u}^T J_\theta \tag{2}$$

To use (2) for calculating $J_\theta \boldsymbol{v}$, we define $g(\boldsymbol{u}) = \boldsymbol{u}^T J_\theta$. Since $\boldsymbol{u}^T J_\theta$ is a vector in $\mathbb{R}^n$, the mapping of $\boldsymbol{v}$ can be defined as a function $g(\boldsymbol{u}) = \boldsymbol{u}^T J_\theta$, where $g(\boldsymbol{u}) : \mathbb{R}^m \to \mathbb{R}^n$. Hence, we can take derivative of $g(\boldsymbol{u})$ with respect to $\boldsymbol{u}$, while providing the left multiplying vector $\boldsymbol{v}$:

$$\boldsymbol{v}^T \frac{\partial g}{\partial \boldsymbol{u}} = \boldsymbol{v}^T \frac{\partial(\boldsymbol{u}^T J_\theta)}{\partial \boldsymbol{u}} \tag{3}$$

$$= \boldsymbol{v}^T {J_\theta}^T \tag{4}$$

$$= (J_\theta \boldsymbol{v})^T \tag{5}$$

In practical implementation, $\boldsymbol{u}$ can be any dummy vector such as the vector of all ones.

3 Deriving Gauss-Newton matrix vector product $G\boldsymbol{v}$

According to notations in [3], the loss function is defined as $\xi(\boldsymbol{z}^{L+1}(\boldsymbol{\theta}))$, where $\boldsymbol{z}^{L+1}$ is the pre-softmax layer, and G equals:

$$G = J^T B J \tag{6}$$

We first take derivative of loss ξ with respect $\boldsymbol{z}^{L+1}$ to obtain $\frac{\partial \xi}{\partial \boldsymbol{z}^{L+1}}$.

Then we calculate the right multiplication of the Jacobian matrix of vector $\frac{\partial \xi}{\partial \boldsymbol{z}^{L+1}}$ by $\boldsymbol{v}$ using the technique from section 1, where $f(\boldsymbol{\theta})$ is substituted with $\frac{\partial \xi}{\partial \boldsymbol{z}^{L+1}}$:

$$\frac{\partial(\xi/\partial \boldsymbol{z}^{L+1})}{\partial \boldsymbol{\theta}^T} \boldsymbol{v} = \frac{\partial^2 \xi}{\partial(\boldsymbol{z}^{L+1})^T \partial \boldsymbol{z}^{L+1}} \cdot \frac{\partial \boldsymbol{z}^{L+1}}{\partial \boldsymbol{\theta}^T} \boldsymbol{v} \tag{7}$$

$$= BJ\boldsymbol{v} \tag{8}$$

Finally we calculate $G\boldsymbol{v}$ by the left multiplication of a Jacobian matrix, where we treat $BJ\boldsymbol{v}$ as the left multiplying vector $\boldsymbol{u}$ and $f(\boldsymbol{\theta})$ as $\boldsymbol{z}^{L+1}$ in equation (2):

$$(BJ\boldsymbol{v})^T \frac{\partial \boldsymbol{z}^{L+1}}{\partial \boldsymbol{\theta}} = (BJ\boldsymbol{v})^T J \tag{9}$$

$$= \boldsymbol{v}^T J^T B J \tag{10}$$

$$= (G\boldsymbol{v})^T \tag{11}$$

References

[1] Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *Journal of Machine Learning Research*, 18(153):1–43, 2018.

[2] Jamie Townsend. A new trick for calculating Jacobian vector products, 2017.

[3] Chien-Chih Wang, Kent Loong Tan, and Chih-Jen Lin. Newton methods for convolutional neural networks. *ACM Transactions on Intelligent Systems and Technology*, 2019. To appear.