Jacobian matrix in PyTorch

Last Updated : 15 Jul, 2021

Introduction:

The Jacobian is a very powerful operator used to calculate the partial derivatives of a given function with respect to its constituent latent variables. For refresher purposes, the Jacobian of a given function $f :^n \to^m$ with respect to a vector $\mathbf{x} = \{x_1, ..., x_n\} \in^n$ is defined as

$$\mathrm{J}_f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} & \cdots & \frac{\partial f}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

Example:

Suppose we have a vector $\mathrm{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$ and a function $\mathrm{f(x)} = \mathrm{f}(\mathrm{x}_1, x_2, x_3) = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix} = \begin{bmatrix} x_1 + x_2 \\ x_1 \times x_3 \\ x_2^3 \end{bmatrix}$. To calculate the Jacobian of f with respect to $\mathbf{x}$, we can use the above-mentioned formula to get

$$J_f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} & \frac{\partial f}{\partial x_3} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} \\ \frac{\partial f_3}{\partial x_1} & \frac{\partial f_3}{\partial x_2} & \frac{\partial f_3}{\partial x_3} \end{bmatrix} = \begin{bmatrix} \frac{\partial(x_1+x_2)}{\partial x_1} & \frac{\partial(x_1+x_2)}{\partial x_2} & \frac{\partial(x_1+x_2)}{\partial x_3} \\ \frac{\partial(x_1\times x_3)}{\partial x_1} & \frac{\partial(x_1\times x_3)}{\partial x_2} & \frac{\partial(x_1\times x_3)}{\partial x_3} \\ \frac{\partial x_2^3}{\partial x_1} & \frac{\partial x_2^3}{\partial x_2} & \frac{\partial x_2^3}{\partial x_3} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 \\ x_3 & 0 & x_1 \\ 0 & 3\times x_2^2 & 0 \end{bmatrix}$$

To achieve the same functionality as above, we can use the **jacobian()** function from Pytorch's **torch.autograd.functional** utility to compute the Jacobian matrix of a given function for some inputs.

> ***Syntax:*** *torch.autograd.functional.jacobian(func, inputs, create_graph=False, strict=False, vectorize=False)*
>
> ***Parameters:***
>
> - ***func:*** *A Python function which takes input and outputs a Pytorch Tensor(or a tuple of Tensors).*
> - ***inputs:*** *The inputs are passed as parameters to the 'func' method. The input can be a single Pytorch Tensor(or a tuple of Tensors)*
> - ***create_graph:*** *If True, the autograd engine creates a backpropable graph for doing further operations on gradients. Defaults to False.*
> - ***strict:*** *If True, an error will be raised when the engine detects that there exists an input such that all the outputs are independent of it. If False, returns zero gradients for such inputs. Defaults to False.*
> - ***vectorize:*** *Still in it's experimental phase if True, the function uses the vmap prototype feature to compute the gradients by only one call of the autograd engine instead of one call per each row of the matrix. Defaults to False.*

Installation:

For this article, you only need the torch utility, which can be downloaded through the pip package manager using:

```
pip install torch
```

Example usage of the function:

using them for representing both the inputs vectors and the given function. This article assumes a basic familiarity with Pytorch tensors which can be quickly reviewed by going through Pytorch articles.

Theoretical verification:

Suppose we have a vector $\mathrm{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 3 \\ 4 \\ 5 \end{bmatrix}$ as a given input. By plugging in the values of $\mathbf{x}$ into the above derived equation we will get

$$\mathrm{J}_f(\mathbf{x}) = \begin{bmatrix} 1 & 1 & 0 \\ x_3 & 0 & x_1 \\ 0 & 3 \times x_2^2 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 \\ 5 & 0 & 3 \\ 0 & 3 \times 4^2 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 \\ 5 & 0 & 3 \\ 0 & 48 & 0 \end{bmatrix}$$

Code: Python implementation to show the working of Jacobian Matrix using Pytorch

Python

```python
from torch.autograd.functional import jacobian
from torch import tensor

#Defining the main function
def f(x1,x2,x3):
    return (x1 + x2, x3*x1, x2**3)

#Defining input tensors
x1 = tensor(3.0)
x2 = tensor(4.0)
x3 = tensor(5.0)
```

Output:

```
((tensor(1.), tensor(1.), tensor(0.)),
(tensor(5.), tensor(0.), tensor(3.)),
(tensor(0.), tensor(48.), tensor(0.)))
```

The output is exactly similar to our theoretical verification! Using a similar approach, we can calculate the Jacobian matrix of any given function using the Pytorch API.

References:

1. https://pytorch.org/docs/stable/autograd.html#torch.autograd.functional.jacobian
2. https://pytorch.org/docs/stable/tensors.html

RECOMMENDED ARTICLES

Page : 1 2 3

01 Python - tensorflow.GradientTape.jacobian()
04, Jul 20

02 Python - Matrix multiplication using Pytorch
19, Jan 21

03 Find determinant of a complex matrix in PyTorch
15, Apr 21

04 Python | PyTorch sin() method
12, Dec 18

05 Install Pytorch on Linux
27, May 21

06 Python | PyTorch sinh() method
12, Dec 18

07 Python | PyTorch cosh() method
12, Dec 18

08 Python | PyTorch tanh() method
12, Dec 18

Article Contributed By :

adityasaini70
@adityasaini70

Vote for difficulty

Easy Normal Medium Hard Expert

Improved By : simmytarika5

Article Tags : Picked, Python Framework, Python-PyTorch, Machine Learning, Python