A Project Report

On

Algorithmic Differentiation

BY

C. S. Adityakrishna

2013A7PS387H

Under the supervision of

Dr N. Anil

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS OF

MATH F376: DESIGN ORIENTED PROJECT

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI (RAJASTHAN)

HYDERABAD CAMPUS

(MARCH 2016)

ACKNOWLEDGMENTS

I would like to thank Dr N. Anil for constant guidance and support. I would also like to thank the Department of Mathematics for providing the opportunity to pursue this project under Dr Anil's supervision.

Birla Institute of Technology and Science-Pilani,

Hyderabad Campus

Certificate

This is to certify that the project report entitled "**Algorithmic Differentiation**" submitted by Mr. C. S. Adityakrishna (ID No. 2013A7PS387H) in partial fulfillment of the requirements of the course MATH F376, Design Oriented Project Course, embodies the work done by him under my supervision and guidance.

Date: 25/3/16 **(Dr N. Anil)**

BITS- Pilani, Hyderabad Campus

ABSTRACT

Algorithmic Differentiation (herein referred to as AD) is a novel approach to computing derivatives of functions. The concept of AD is by no means recent[1]. Firstly, we disucss some of the traditional approaches, including Symbolic Differentiation, Method of Finite Differences (Numerical Differentiation) and CTSE, along with their respective advantages and disadvantages. Further, we provide rationale as to why Algorithmic Differentiation remains the best of all these approaches. We then delve into the black box that is AD and highlight two of the fundamental methods of applying AD, i.e, the Forwarde Mode and the Reverse Mode. We discuss properties of both these methods and evaluate the crietria under which one mode operates better than the other. Finally, we look at some finer computational and implementation details and look at how the AD technique can be fine tuned for the purpose of Machine Learning.

CONTENTS

Title page…….1

Acknowledgements……..2

Certificate…..……....3

Abstract……....4

Traditional Approaches...…….……....6

Algorithmic Differentiation...……...7

Implementation Techniques.……10

Future Plan..……11

References……12

TRADITIONAL APPROACHES

Symbolic Differentiation

In this method, analytical derivates are obtained by deriving the function expression as a person would. The program reads and understands the function expression and produces corresponding derivative code. This approach breaks down when the function expression requies the usage of loops and branches.

Method of Finite Differences

Simplistically, this method involves calculating the difference coefficients of functions around given point *h*, by calculating the limit of *h* tending to zero. While there are several numerical analysis improvements to this basic idea, this approach is prone to several types of calculation and computational(precision) errors. This method also involves high computational cost as it requires a function evaluation for every dependant variable in the system.

$$\frac{\partial l}{\partial \alpha_i} = \frac{l(\alpha_i + \Delta\alpha_i) - l(\alpha_i - \Delta\alpha_i)}{2\Delta\alpha_i} + O(\Delta\alpha_i)^2$$

Complex Taylor Series Expansion

The Complex Taylor Series Expansion method uses an imaginary perturbation of $i\Delta a$. The Taylor Series expansion of the perturbed function is used to determine the derivative. This method moves the issue of cancellation errors as seen while calculating Numerical Derivatives, but is still computationaly very expensive.

ALGORITHMIC DIFFERENTIATION

AD works on the principle of Chain Rule at the operator level. This is similar to symbolic differentiation, however, the derivative function code is not explicitly generated by the AD tool. The AD tool will interpret the input program as a sequence of elementary operations along with some intrinsic functions. The tool will then output a derivative program that is obtained by applying chain rule sequentially to each of the elementary operations. Methods of implementing (in code) and complexity of these transformations are discussed in later sections.

Consider the function,

$$f = f_1(f_2(f_3(...(f_n(x)))))$$

$$\frac{\partial f}{\partial x} = \frac{\partial f_1}{\partial f_2} * \frac{\partial f_2}{\partial f_3} * \frac{\partial f_3}{\partial f_n} ... * \frac{\partial f_n}{\partial x}$$

In Forward Mode, the chain rule propagates from right to left. i.e. the inner most function to outermost. In Reverse Mode, the direction of propagation is from left to right. However it must be noted that Forward and Reverse Modes are just two (extreme) ways of traversing the chain rule. The problem of computing a full Jacobian of f: $\mathbb{R}^n \rightarrow \mathbb{R}^m$ with a minimum number of arithmetic operations is known as the *optimal Jacobian accumulation* (OJA) problem, which is NP-complete under mild assumptions. This is central to our reasoning that AD Tools can be tailored for the purposes of training arbitrarily designed Neural Networks by virtue of the basic architecture that is common to all networks.

The propagation of the chain rule is best illustrated via a computational graph. There exists multiple possible computational graphs for a single given expression (function) by rearranging the associativity of the operations or by treating repeated sub-expressions as unique. We demonstrate both Modes with a toy example:

Consider the function,

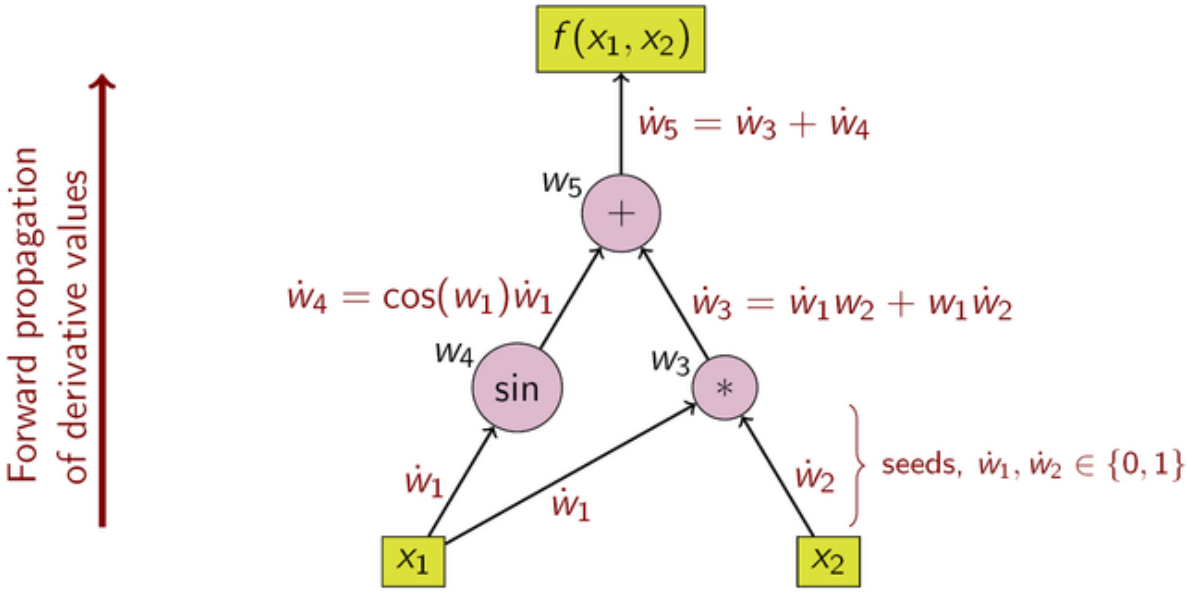
$$f(x_1, x_2) = \sin(x_1) + x_1 \cdot x_2$$

We need to compute,

$$\frac{\partial f}{\partial x_1} \quad \frac{\partial f}{\partial x_2}$$

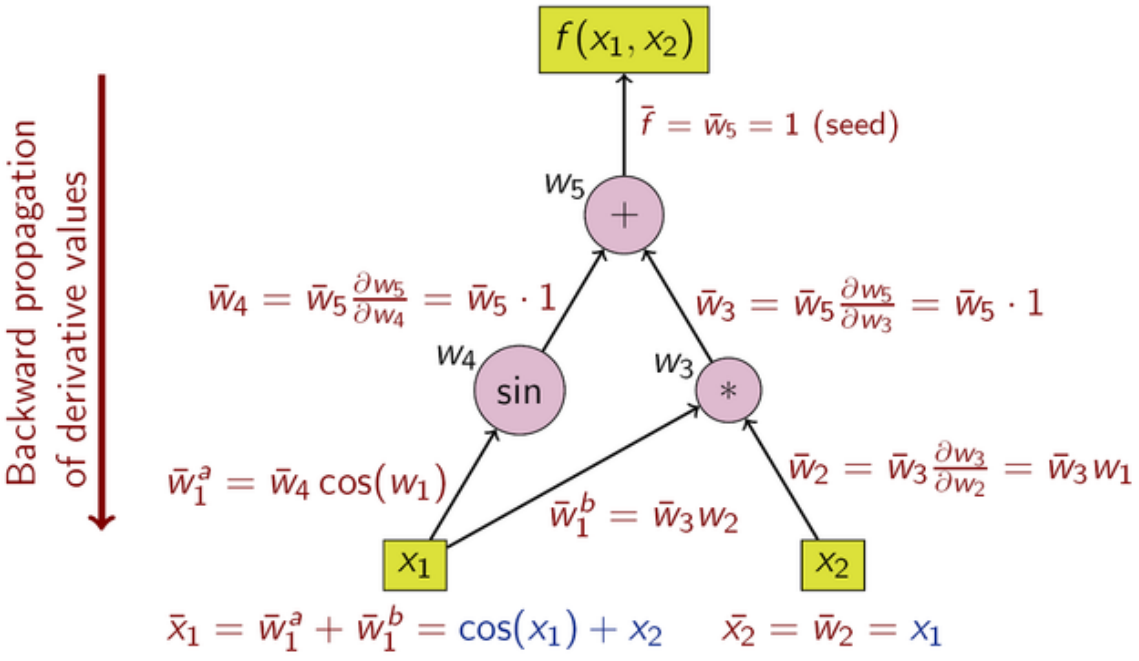
Forward Mode

In Forward Mode, a forward (evaluative pass) is required for each independant variable. For each such variable, it is initialised to 1 and the remaining variables to 0.



Reverse Mode

In Reverse Mode, a single forward evaluation is performed after which a backward pass provides the derivative of the output function with respect to *all* the input variables. While this seems magical, it is at the cost of high memory requirement. Every intermediate variable must be stored during the forward pass for purpose of evaluating the derivative.



Comparison of Modes

For any function $f: \mathbb{R}^n \rightarrow \mathbb{R}^m$

Choosing the optimal technique for computing the derivative of requires consideration of computational complexity and memory requirements. With these criteria in mind a simple thumb rule comes to surface.

Forward Mode requires ***n*** passes to evaluate the derivative whereas Reverse Mode requires ***m*** passes. Therefore, for m >> n, Reverse Mode is preferred and Forward Mode for n >> m. It sohuld also noted that Reverse Mode *may* require high memory storage as all intermediate variable in the forward pass need to be stored on a tape. Some methods like checkpointing help reduce the memory requirements for the process but Reverse Mode is still memory intensive.

In terms of complexity, we can place some bounds on the running time of the appended/modified function evaluation code. Reverse method provides a complete set of derivatives for a cost of between one and four function evaluations. As mentioned earlier, determining the OJA is an NP-complete problem.

IMPLEMENTATION TECHNIQUES

There two primary implementation strategies, the first is called Soure Code Transformation and the second, Operator Overloading.

Source Code Transformation

AD Tool would pre-process the function written in specified language. The tool would then append/modify the code accordingly and the forward to the compiler. This allows the compiler to perform compile time optimizations as the compiler is unaware of the intermediate pre-processing.

Operator Overloading

Overloaded behaviour for every elemetary operation and intrinsic function is fed to the compiler along with the original code. The derivative code is interleaved with the function evaluation code. This approach requires redefinition/restructuring of some basic data types to allow for this overloaded behaviour.

Existing Libraries

Several libraries are present that provide a variety of features via several implementation techniques which stem from the two approaches discussed above. Some of these libraries written for C/C++ are-

1. Tapenade (SCT)
2. Adept (OO)
3. Stan (OO)

Some popular libraries written for other languages are-

1. Autograd (Python/OO)
2. Theano (Python/OO)
3. Torch (LuaJIT/OO)

FUTURE PLAN

Neural Networks

The basic framework used in Artificial Neural Networks (ANNs) provides some headway in optimizing the computational graphs that are generated by general purpose AD tools.

The standard architecture of Neural Networks consists of an Input Layer followed by multiple Hidden Layers and finally an Output Layer. In a feedforward evaluation of the network (function), subsequent Hidden Layers are computed as a summation of the product of weights and activation values from the previous layer. Once the activations for every layer is computed, an activation function is applied and then activation for the next layer are computed.

These activation functions can be treated as intrinsic functions that are abstracted while generating the Computational Graph. Some common activation functions are the logistic function, sigmoid function, Rectified Linear Unit, tanh etc.

The performace of the network is measured by a variety of cost functions including Mean Squared Error, Softmax Activation, Cross-Entropy Loss etc.

With these facts in mind, we would like to experiment on the various techniques of using forward/reverse mode, operator overloading/source code transformation, to concretely show the efficiency of a new tool for ANNs over general purpose AD tools currently available.

REFERENCES

Evaluating Derivatives – Andreas Griewank

Automatic differentiation in machine learning: a survey – Baydin et al.

https://en.wikipedia.org/wiki/Automatic_differentiation

https://en.wikipedia.org/wiki/Artificial_neural_network