Automatic differentiation in machine learning: a survey (2018)

Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, Jeffrey Mark Siskind

This article makes a general survey of AD techniques, implementations , usages with a link to various ML areas . The author believes that general-purpose AD is the future of gradient based ML techniques . An important direction in the future research is nested AD techniques in ML .

Derivation techniques Overview:

There are four main classes of derivation techniques in computer programs :
(Also some of their problems that were addressed in AD)

1) Manual derivatives coding
   (Require closed form expression , problematic in programming flow control constructions).
2) Numerical differentiation
   (inherently ill-conditioned and unstable) , O(n) complexity
3) Symbolic differentiation
   (expression swell problem)
4) AD
   AD refers to specific techniques of computing derivatives through accumulating values during the program execution , and generating derivative value (not expression) . It is applicable to regular code (loops branching etc) , and not only to arithmetic expressions .Backpropagation can be considered as a particular case of AD .

AD overview:

Non standard program interpretation when standard flow is augmented with derivatives computation .

Forward mode (FM):

Together with forward primary computation trace we generate a corresponding tangent trace , by applying a chain rule and generating derivative trace .

Then in order to compute the derivative for each output variable with respect to the input vector *x*, we traverse the tangent trace for each input value x_i . FM complexity depends on input parameters number .

First we associate an initial derivative for each x_i , which is set to one while all other initial derivatives are set to zero when we compute the derivative for this particular x_i . Evaluating the tangent trace with particular input $x=a$ provides one column of Jacobian matrix calculated at point a.

In addition by initializing the initial derivatives to any number (other than one) we can compute Jacobian vector product ***J_y*r*** easily .

Mathematically , FM can be viewed as evaluating the function using dual numbers .

Revese mode (RM):

AD in the RM corresponds to generalized backpropogation technique. As opposed to the intermediate derivative variable *v* in tangent trace in FM which represents sensitivity of the *v* to input changes . In RM it represents the sensitivity of the output to changes in v .

RM mode is a two phase process . At the first phase the original function runs forward populating the intermediate variables and recording the dependencies in a computational graph . In the second phase the derivative is computed by propagating the adjoints *v* from the output to the input . For each intermediate variable *v* we check what variables it affects output through (that their derivative was already computed) , and compute its derivative by chain rule and sum.

As opposed to FM we traverse the resulting graph back for each output variable , producing the rows of the Jacobian matrix . And like in FM we can easily compute Jacobian transpose multiplication (***J_y)^T*r .*** RM complexity depends on output vector size .

RM comes also with increased storage requirements

AD usage:

The paper mentios AD and gradient based optimizations in general , AD and CV, NLP , Deep learning , differential programming . There is also a paragraph that summarizes current and possible usage of AD in my interest area (Probabilistic modeling and inference) .

{This sections mention general bindings of AD with these areas , mention important papers and works - may be useful }

AD Implementations:

A principle consideration in any AD is the performance overhead of AD arithmetics and bookkeeping . “Perturbation confusion” bugs class and major numerical issues that can be highly relevant to AD are mentioned .

A taxonomy of implementation techniques is *elemental,operator overloading , compiler-based and hybrid* methods .

Elemental : Replacing elementary operations manually (math operators) by AD-enabled library .

Compiler-based : Automated decomposition to AD enabled code .The code can be written using language extensions , which is converted to regular language code by preprocessing. There can be new languages with integrated AD capabilities .

Operator overloading : Redefining the semantics of the operators .