Study Session

Topic(Automatic differentiation)
04/08/2017

Motivation

1. Sharing my own project had been one of my dutyness in this 2 weeks

2. Making you fun intellectually \Ö/ (never overwhelming or aligned)

3. Making things much easier (not repeating any web source about this topic)

4. Providing tips to read DL library (e.x. Tensorflow) ? :@ ?

Computation Graph (one of my slides of my last talk)

Derivative of objective variables with respect to each single variables

Derivative of parent -node with respect to child-node (child node : variable)
combine whole generations using chain rule

Let - +. , / *

Operator

case + | - -> d(P) / d(C) = 1

case operator * -> d(P) / d(C) = the value of a sibling

Factorization reduce numbers of branches retaining same calculations

Computation Graph (one of my slides of my last talk)

Derivative of objective variables with respect to each single variables

Derivative of parent -node with respect to child-node (child node : variable)
combine whole generations using chain rule

Too much Information on 1 slide

Let - +. , / *

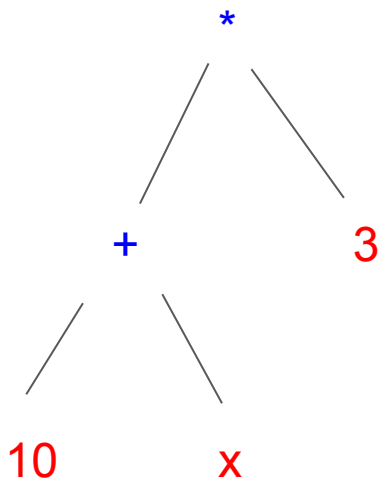
Operator

case + | - -> d(P) / d(C) = 1

case operator * -> d(P) / d(C) = the value of a sibling

Factorization reduce numbers of branches retaining same calculations

Representation of calculation



f(x) = 10 + x * 3

Leaf : Var or Number

Intermediate Node : Operator

Tree representation on Haskell

```
data S = M Lp Op S S Rp
       | N Nun
       | V Var
       deriving (Show)

data Op = Op Char deriving (Show)
data Lp = Lp Char deriving (Show)
data Rp = Rp Char deriving (Show)
data Nun = Nun Float deriving (Show)
data Var = Var Char deriving (Show)
data Nn = Int | Float | Double deriving (Show)
```

How to calculate derivatives

1. Bottom Up & non-Symbolic

2. Top Down & non-Symbolic

3. Bottom Up & Symbolic

4. **Top Down & Symbolic <- today’ s talk**

Basic Idea

- Set multiple rules with respect to a combination of an operator and a pair of values on child nodes

- Let degree of every differential variable of 1, treat them as a separate variable

```
ad  :: Char -> Bool -> (S,[S]) -> S -> [S]     :arguments  :return type
```

```
ad  :: Char -> Bool -> (S,[S]) -> S -> [S]
```

: differential variable

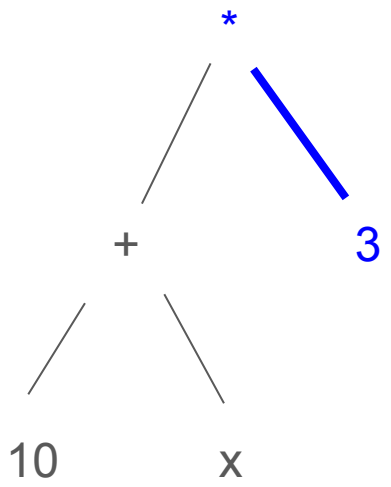
: States is (empty / set)

: Tuple which consists of State & Intermediate answer

: List of tree (return)

Examples

Example1 (multiplication && non-variable)

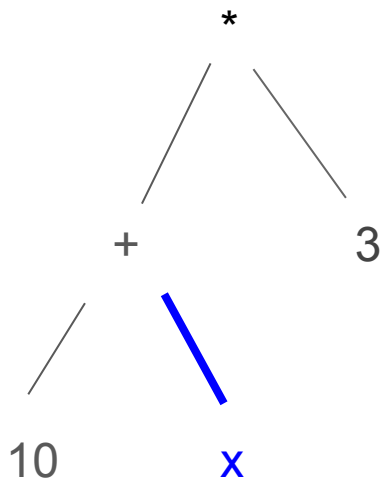


f(x) = 10 + x * 3

f'(x) = 3

=> Store states(3) and dive deeper with it

Example1 (addition && variable)

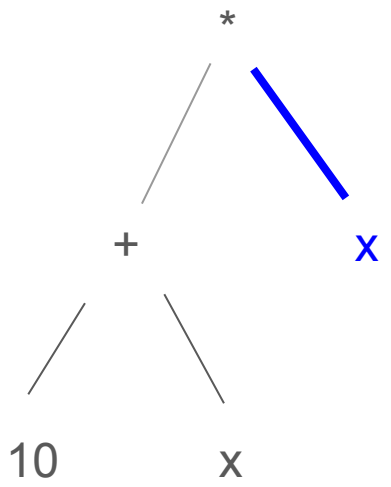


f(x) = 10 + x * 3

f'(x) = 3

=> set (Num 1) on an answer list

Example2 (multiplication && variable)



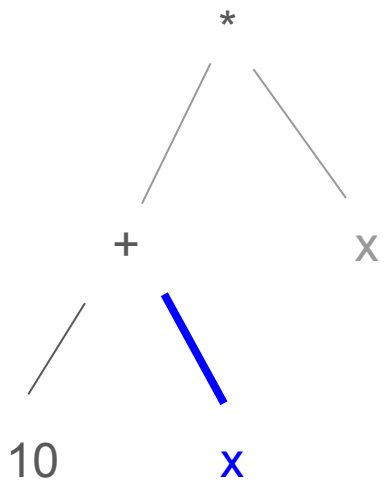
f(x) = (10 + x) * x

f’(x) = 2 x + 10

=> Store States(x) and dive deeper

Set (10 + x) in an answer list

Example2 (addition && variable)



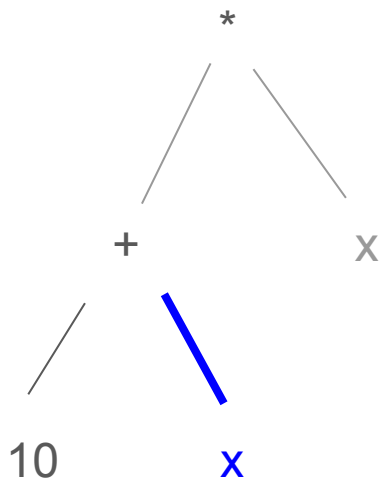
f(x) = (10 + x) * x

f'(x) = 2 x + 10

=> Merge (Num 1) with kept state

(x) and put it on an Answer list

Example2



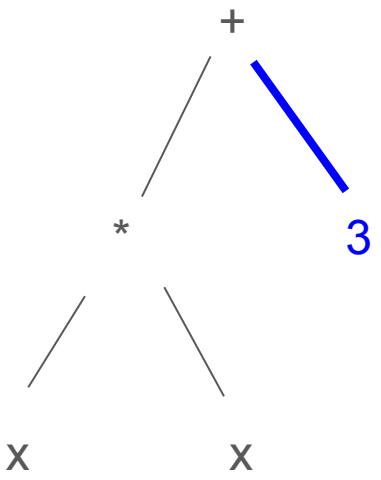
f(x) = (10 + x) * x

f'(x) = 2 x + 10

=> after coming up, you will see the list of

Tree ; [x,x+10]

Example3 (addition & non-variable)

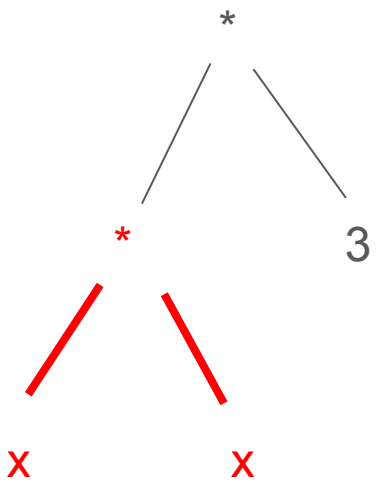


f(x) = x * x + 3

f’(x) = 2x

=> let it go and move on

Example3 (multiplication & variable)



f(x) = x * x + 3

f’(x) = 2x

=> Take the other side and put it on an

Answer box [x,x]

Examples End :o

How many rules do you need?

(+) Var Var

(+) Var else

(+) else Var

(+) else else

(*) Var Var

(*) Var else

(*) else Var

(*) else else

For conclusion, check Ad.hs out

Is that all?

You need another 1.

No match will return empty list. Ö

Top Down Approach

RULE

<Down>

1. Multiplication & Variable -> Put an opposite side as a state
2. Addition & variable -> Add 1 and go deeper

- Tail Recursion (= having a state as an argument)
- Integration of each node when comes up

Bottom up Approach

1. Put an unique name to each single differential variable
2. Recursion from bottom
3. Put an original name to

Good : Intuitive, normal recursion

Difficult : how to make a line between returning value and node

Dirty Side of this code

- What is an Initial condition of state ??

  -> resolved introducing another bool argument

  -> should use state monad

- Rules should not be defined as a set of functions

  -> should be provided as an algebraic data type having instance of Monoid

Conclusion

- Less than 30 lines of code (8 rules) is in charge of automatic differentiation.

  => Simple ö

- Coding is much less important than thinking

Fun with syntax analytics (Supplement)

- Every programming language is defined by finite set of rules which consists of

  nonterminal / terminal signature.

- Writing programming is sequential tree construction which has previous state and proceeding states.
- How do you evaluate your code (tree) itself given an objective? How do you update it?

  (scenario :: programming learning service)

Question? Ü

Bye !