Neural Networks are Program, too.

Marisa Kirisame

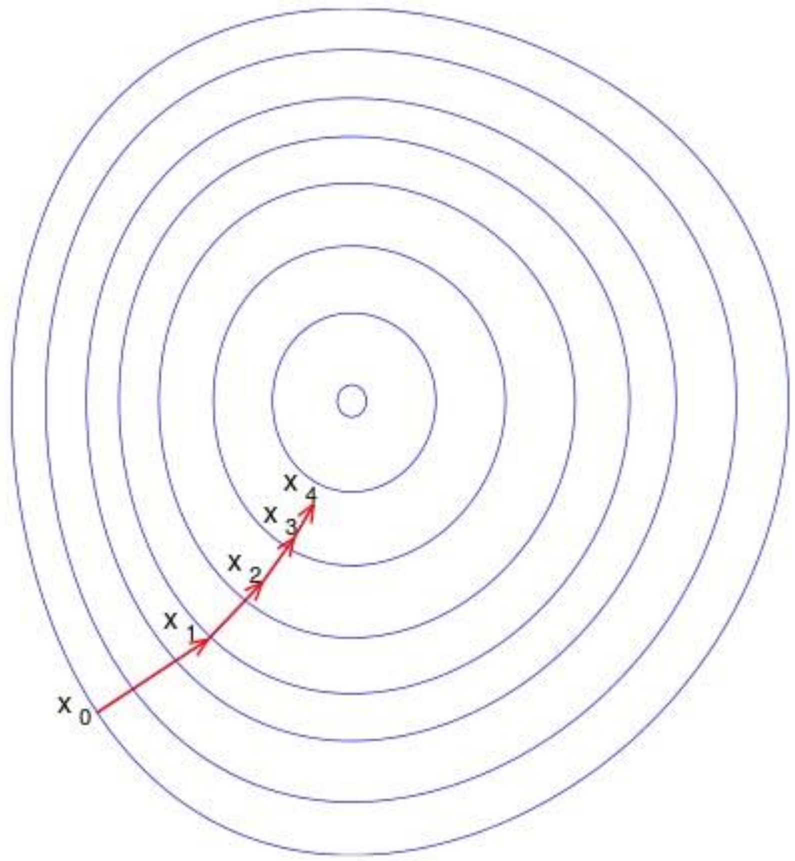
University of Washington

Agenda

- Neural Networks are Program
- Apply PL/FP to Neural Network(NN)

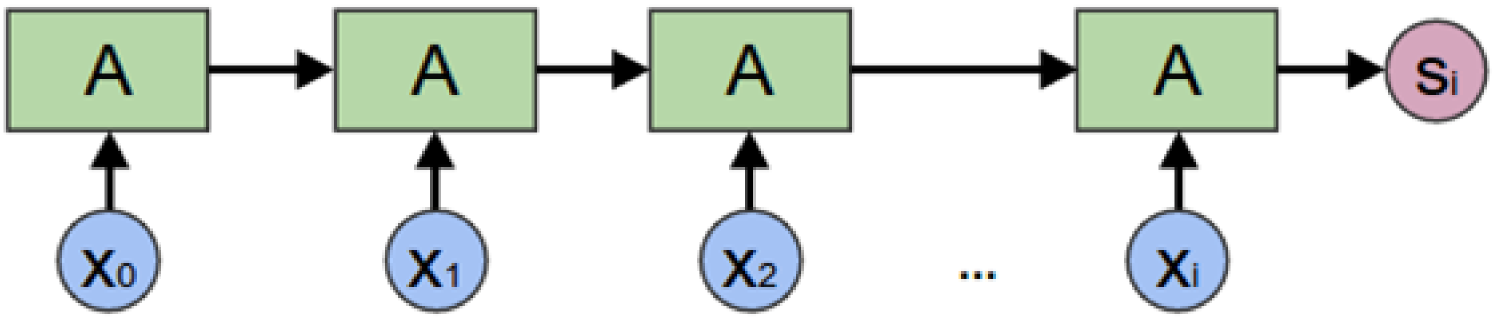
What is a Neural Network?

- A program that contains implicit parameters (weights).
- Find Best Weight
- https://en.wikipedia.org/wiki/Gradient_descent



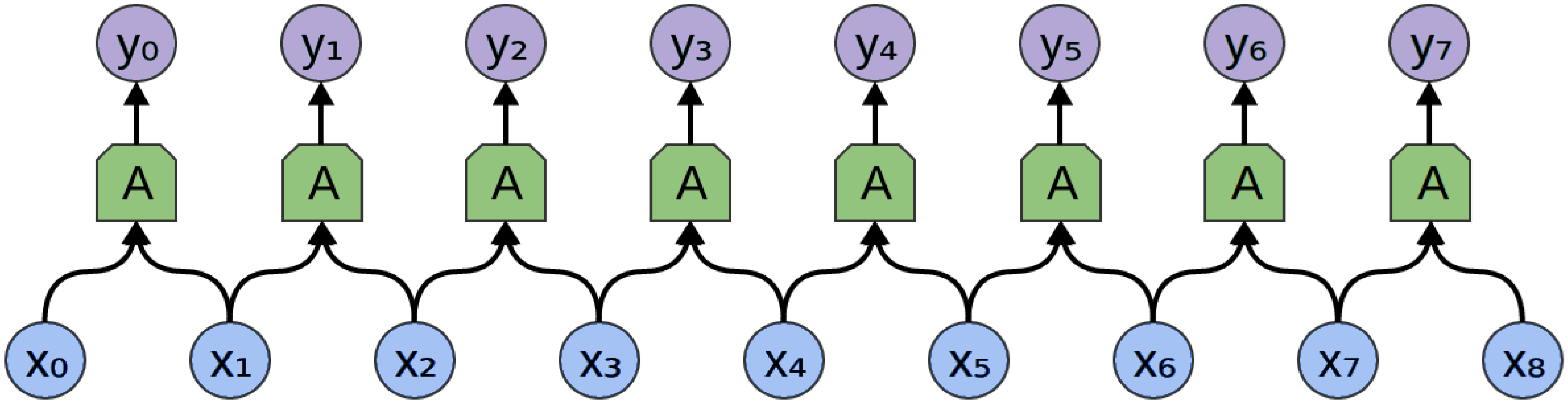
Quick Example

- http://colah.github.io/posts/2015-09-NN-Types-FP/



More Examples

- http://colah.github.io/posts/2015-09-NN-Types-FP/
- http://blog.emillon.org/posts/2012-10-18-comonadic-life.html



How to Train Your Neural Network

- train: **w** → **NN** → **Input** → **Output** → **w**
- train *w nn inp out* =

  let

    res = *nn w inp*

    loss = ½ * (*res* – *out*) ^ 2

    dloss = (*res* – *out*) * d(*res*)/d*w*

  in *w* – *dloss*

Roadmap

- Neural Network
- **Naive Automatic Differentiation(AD) – Quadratic**
- Forward Mode AD
- Derivative of Multiple Variable
- Derivative of More Variable
- Impl NN
- Meaning of AD on PL

Begging the question

- Use high school calculus rule
- $\frac{dx}{dx} = 1$
- $\frac{dy}{dx} = 0$
- $\frac{d\left(f(x) + g(x)\right)}{dx} = \frac{d\left(f(x)\right)}{dx} + \frac{d\left(g(x)\right)}{dx}$
- $\frac{d\left(f(x) \cdot g(x)\right)}{dx} = \frac{d\left(f(x)\right)}{dx} \cdot g(x) + \frac{d\left(g(x)\right)}{dx} \cdot f(x)$
- $\frac{d\left(f\left(g(x)\right)\right)}{dx} = \frac{d\left(f(x)\right)}{dx} \cdot g(x) \cdot \frac{d\left(g(x)\right)}{dx}$

How to Train Your Neural Network 2

Solve for $x^2 + 2x + 3 = 27$ (x > 0)

train: $x \rightarrow x$

train x =

let

res = x * x + 2 * x + 3

loss = ½ * (*res* – 27) ^ 2

dloss = (*res* – 27) * (2 * x + 2)

in x – *dloss*

Too Young Too Simple

- Naive approach doesn't scale!
- Consider $\frac{d\left(f(x)g(x)h(x)\right)}{dx}$
- $f'(x)g(x)h(x) + f(x)g'(x)h(x) + f(x)g(x)h'(x)$

Roadmap

- Neural Network
- Naive Automatic Differentiation(AD) – Quadratic
- **Forward Mode AD**
- Derivative of Multiple Variable
- Derivative of More Variable
- Impl NN
- Meaning of AD on PL

Simple Sharing Strategy

- Interpret expression as a pair

lit *r*	→ (lit *r*, lit 0)
(*l*, *ldiff*) + (*r*, *rdiff*)	→ (*l* + *r*, *ldiff* + *rdiff*)
(*l*, *ldiff*) * (*r*, *rdiff*)	→ (*l* * *r*, *l* * *rdiff* + *r* * *ldiff*)

- Now linear time
- Forward Mode Automatic Differentiation(AD)

Example

$x * x + 2 * x + 3$

$\rightarrow (x, 1) * (x, 1) + (2, 0) * (x, 1) + (3, 0)$ [lit $r \rightarrow$ (lit r, lit 0)]

$= \ (x * x, 2 * x) + (2, 0) * (x, 1) + (3, 0)$ $[(l, ldiff) * (r, rdiff) \ \rightarrow (l * r, l * rdiff + r * ldiff)]$

$= \ (x * x, 2 * x) + (2 * x, 2) + (3, 0)$ $[(l, ldiff) + (r, rdiff) \ \rightarrow (l + r, ldiff + rdiff)]$

$= \ (x * x + 2 * x, 2 * x + 2) + (3, 0)$

$= \ (x * x + 2 * x + 3, 2\ x + 2)$

How to Train Your Neural Network 3

Solve for *x* ^ 2 + 2 *x* + 3 = 27

train: x → x

train *x* =

let

res = (*x*, 1) * (*x*, 1) + (2, 0) * (*x*, 1) + (3, 0)

loss = (½, 0) * square(*res* – (27, 0))

--loss = (½ * square(x * x + 2 x + 3), (res – 27) * (2x + 2))

in *x* – rhs *loss*

Remember the Talk Title?

- We don't want it to work on simple arithmetic.
- We want it to work on program.

Adding static typing

- **type** Real = Double
- **type family** DiffType (*x* : *) : *
- **type instance** DiffType (*a* → *b*) = DiffType *a* → DiffType *b*
- **type instance** DiffType (*a* + *b*) = DiffType *a* + DiffType *b*
- **type instance** DiffType (*a* * *b*) = DiffType *a* * DiffType *b*
- **type instance** DiffType Real = (Real * Real)

Looking back

- We achieve the closure property
- Do stuff with AST
- Everything is typed

Roadmap

- Neural Network
- Naive Automatic Differentiation(AD) – Quadratic
- Forward Mode AD
- **Derivative of Multiple Variable**
- Derivative of More Variable
- Impl NN
- Meaning of AD on PL

Differentiating wrt multiple variable?

- Look carefully at the transformation
- **lit** *x* → (**lit** *x*, **lit** 0)
- \+ → **λ**(*l*, *ld*) (*r*, *rd*) → (*l* + *r*, *ld* + *rd*)
- \- → **λ**(*l*, *ld*) (*r*, *rd*) → (*l* – *r*, *ld* – *rd*)
- * → **λ**(*l*, *ld*) (*r*, *rd*) → (*l* * *r*, *l* * *rd* + *r* * *ld*)
- / → **λ**(*l*, *ld*) (*r*, *rd*) → (*l* / *r*, *l* / *r* * *ld* – *l* / (*r* * *r*) * *rd*)
- exp → **λ**(*x*, *xd*) → **let** *ex* = exp *x* **in** (*ex*, *ex* * *xd*)

Vector Space!

- Unit is a Vector Space, 0 weight
- **R** is a Vector Space, 1 weight
- (*vl* * *vr*) is a Vector Space, have added weight
- *V*[1000] is a Vector Space
- Minimal definition:
- **0** :: *v*
- **+** :: $v \rightarrow v \rightarrow v$
- **scale** :: **R** $\rightarrow v \rightarrow v$
- DiffType now take an extra parameter *v*, to represent the vector space

Example

$x * x + y * y + x * y$

$\rightarrow (x, (1, 0)) * (x, (1, 0)) + (y, (0, 1)) * (y, (0, 1)) + (x, (1, 0)) * (y, (0, 1))$

$= \quad (x * x, (2 * x, 0)) + (y * y, (0, 2 * y)) + (x * y, (y, x))$

$= \quad (x * x + y * y + x * y, (2 * x + y, 2 * y + x))$

lit x $\rightarrow$ (**lit** x, **lit** 0)

$+$ $\rightarrow \boldsymbol{\lambda}(l, ld)\ (r, rd) \rightarrow (l + r, ld + rd)$

$*$ $\rightarrow \boldsymbol{\lambda}(l, ld)\ (r, rd) \rightarrow (l * r, l * rd + r * ld)$

Roadmap

- Neural Network
- Naive Automatic Differentiation(AD) – Quadratic
- Forward Mode AD
- Derivative of Multiple Variable
- **Derivative of More Variable**
- Impl NN
- Meaning of AD on PL

A slight problem

- Suppose *v* is Real[10000]
- Expensive to **0**/+/scale
- Accumulate the “scale factor” in a parameter

instance Vector *x* ⇒ Vector (Real → *x*) **where**

  0 = const **0**

  (*l* + *r*) *x* = (*l x* + *r x*)

  (*l* **\`scale\`** *r*) *x* = *r* (*l* * *x*)

Boom!

- Exponential, now much worse
- Consider $b = (a + a)$; $c = (b + b)$; $c + c$
- We need to share the actual function as well
- Cant compare function (Halting Problem)
- But we can compare AST

Iso-Recursive Type

- DiffType v (Fix f) = DiffWrapper [v] f (Fix f)
- DiffType v (DiffWrapper a x) = DiffWrapper (v :: a) x
- Wrapper for a term on (fold on DiffType and x)
- Data Type A La Carte(DTALC) → ADT

Selecting Weight

Basis Unit = Void

Basis **R** = Unit

Basis (*l* * *r*) = Basis *l* + Basis *r*

FreeVector *b* = *b* → Real

FreeVectorBuilder *b* = Map *b* Real

Term Algebra

TermVector *b* = Zero | Basis *b* |

Plus (TermVector *b*) (TermVector *b*) |

Scale Real (TermVector *b*)

TermVectorF *b f* = Zero | Basis *b* | Plus *f f* | Scale Real *f*

TermVector *b* = Fix (TermVectorF *b*)

TermVector is a Vector

- Just call the constructor.
- Still exponential in that example.
- But we reduce the problem to a simpler one.

Hash Consing

- Implementing Explicit and Finding Implicit Sharing in Embedded DSLs.
- State (Bimap (TermVectorF *b* Int) Int) Int

Forward Mode = Backward Mode

- State (Bimap (TermVectorF *b* Int) Int)
- Insert empty bimap, get a pair of <u>bimap</u> and <u>int</u>
- Create Map Int Real
- Map each AST to 'accumulating scaling value‘ - sensitivity
- Wengert List.
- Recurse starting from nodeid, −1 every time
- Return State (Map Int Real) (FreeVectorBuilder *b*)

Backward Mode = Back Propagation

- Match on (TermVectorF *b* Int) from the map:
- Zero — return zero
- Basis *b* — return it
- *l* `plus` *r* — get sensitivity, add to sensitivity of *l* / *r*
- *l* `scale` *r* — get sensitivity, scale *l*, add to *r*
- Recurse and add the two Builders
- Turn Builder into FreeVector

Roadmap

- Neural Network
- Naive Automatic Differentiation(AD) – Quadratic
- Forward Mode AD
- Derivative of Multiple Variable
- Derivative of More Variable
- **Impl NN**
- Meaning of AD on PL

Finally, Neural Network

- A NN of type x is just an $\exists w$. Term ($w \to x$)
- A Term is a NN: $\exists$Unit
- Term ($x \to y$) $\to$ Term $x \to$ Term y
- Finally Tagless

Xor Network

- weight : NN Real
- sigmoid x = 1 / (1 + (exp (− x)))
- add, bias, scale
- neuron : NN ((Real * Real) → Real)
- xor : NN (Real → Real → Real)
- loss : NN ((Real → Real → Real) → Real)
- loss xor : NN Real

Roadmap

- Neural Network
- Naive Automatic Differentiation(AD) – Quadratic
- Forward Mode AD
- Derivative of Multiple Variable
- Derivative of More Variable
- Impl NN
- **Meaning of AD on PL**

Meaning of Differentiating Higher Order Function

- We need to step back
- Denote to nonstandard language Term, NonStdTerm
- Same as the old STLC:

old **lit**: Real → Term Real || **lit**: (Real → Real) → Term Real

lit *x* → **lit** (_ → *x*)

\+ → +, with operational semantic of

lit *l* + lit *r* → lit (*x* → *l x* + *r x*)

Back to The Problem

LR: **forall** *t*. NonStdTerm (DiffType *t*) → **Prop**

LR (*l* * *r*) *t* = **exists** *x y*. *t* <u>evaluates to</u> app (app (mkProd *x*) *y*),
and LR *l x* **and** LR *r y*

LR (*l* + *r*) *t* = **exists** *x*. *t* <u>evaluates to</u> app **left** *x*, LR *l x*
or, exists *y*. *t* <u>evaluates to</u> app **right** *y*, LR *r y*

LR (*l* → *r*) *t* = **forall** *inp*. LR *l inp* → LR *r* (app *t inp*)

LR Unit t = t <u>evaluates to</u> mkUnit

LR Void t = False

The Main Part

LR Real *t*:

t evaluates to (lit orig, lit diff)

forall *x*. diff *x* = (newton differental of orig) on *x*

Denotational Semantics

LR (Real → Real)

Back to Standard

stdify: **forall** *t*, Real → NonStdTerm *t* → Term *t*

app (stdify *r f*) (stdify *r x*) = stdify *r* (app *f x*)

<u>apply</u> (*x* → *x*, *x* → 1) <u>to</u> Real → Real, <u>apply</u> *x* <u>to</u> rhs

= <u>apply</u> (*x*, 1) <u>to</u> Real → Real, <u>take the</u> rhs

- Optimize away the function

Wrapping Up

- deq relate Term *t* and Term (DiffType *t*)
- Main theorem:

  forall *t* (*term*: Term *t*), deq *t* *term* (diff *term*) ∧ LR *t* (diff *term*)

- Logical Relation
- Operational Semantic
- Denotational Semantic

Drawing the Connection (Conclusion)

Data Structure → Forward Mode AD

Semantic/Logical Relation → Meaning of AD

F Algebra, DataKinds → ADT

TypeClass → Generalized Forward AD

Term Algebra(DSL), Hash Consing → Backward AD

DTALC/FTG → typed, extensible framework

Higher Order Function → RNN(fold)

Partial Evaluation → Optimization (Tensorflow fold)

Program Synthesis → Optimization (Latte)

Existential Type → Neural Network

HOAS → Pretty API

Monad → State

Implementation Detail

- Finally Tagless mode, generalized AD, NN
- And more! (Infinite tower of diff, example)
- https://github.com/ThoughtWorksInc/DeepDarkFantasy/
- Example at DDF/Poly.lhs, DDF/Xor.lhs

Citation

- Implementing Explicit and Finding Implicit Sharing in Embedded DSLs
- http://colah.github.io/posts/2015-09-NN-Types-FP/
- Reverse-Mode AD in a Functional Framework

Acknowledgements

- Belleve Invis
- Bill Zorn
- Sam Elliott
- Zachary Tatlock
- Zheng Yang

Question?