Introduction to Deep Learning

Automatic Differentiation

Andres Mendez-Vazquez

June 13, 2025

Outline

Cinvestav

A Historical Perspective

The idea of a Graph Structure was proposed by Raul Rojas

- “Neural Networks - A Systematic Introduction” by Raul Rojas in **1996...**

A Historical Perspective

The idea of a Graph Structure was proposed by Raul Rojas

- "Neural Networks - A Systematic Introduction" by Raul Rojas in **1996...**

TensorFlow was initially released in November 9, 2015

- Originally an inception of the project "Google Brain" (Circa 2011)

A Historical Perspective

The idea of a Graph Structure was proposed by Raul Rojas

- "Neural Networks - A Systematic Introduction" by Raul Rojas in **1996...**

TensorFlow was initially released in November 9, 2015

- Originally an inception of the project "Google Brain" (Circa 2011)
- So TensorFlow started around 2012-2013 with internal development and DNNResearch's code (Hinton's Company)

A Historical Perspective

The idea of a Graph Structure was proposed by Raul Rojas

- "Neural Networks - A Systematic Introduction" by Raul Rojas in **1996...**

TensorFlow was initially released in November 9, 2015

- Originally an inception of the project "Google Brain" (Circa 2011)
- So TensorFlow started around 2012-2013 with internal development and DNNResearch's code (Hinton's Company)

However, the graph idea was introduced in 2002 in torch, the basis of Pytorch (Circa 2016)

- One of the creators, Samy Bengio, is the brother of Joshua Bengio [1]

Outline

Cinvestav

Backpropagation a little brother of Automatic Differentiation (AD)

We have a crude way to obtain derivatives [2, 3, 4, 5]

$$D_{+h}f(x) \approx \frac{f(x+h) - f(x)}{2h} \text{ or } D_{\mp h}f(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$

Cinvestav

If h is small

- then cancellation error reduces the number of significant figures in $D_{+h}f\left(x\right)$.

If h is small

- then cancellation error reduces the number of significant figures in $D_{+h}f(x)$.

if h is not small

- then truncation errors (terms such as $h^2 f'''(x)$) become significant.

Cinvestav

If h is small

- then cancellation error reduces the number of significant figures in $D_{+h}f(x)$.

if h is not small

- then truncation errors (terms such as $h^2 f'''(x)$) become significant.

Even if h is optimally chosen

- the values of $D_{+h}f(x)$ and $D_{\mp h}f(x)$ will be accurate to only about $\frac{1}{2}$ or $\frac{2}{3}$ of the significant digits of f.

Cinvestav

Outline

Cinvestav

Outline

Cinvestav

Avoiding Truncation Errors

We have that

- Algorithmic differentiation does not incur truncation errors.

Avoiding Truncation Errors

We have that

- Algorithmic differentiation does not incur truncation errors.

For example

$$f(x) = \sum_{i=1}^{n} x_i^2 \text{ at } x_i = i \text{ for } i = 1 \dots n$$

Avoiding Truncation Errors

We have that

- Algorithmic differentiation does not incur truncation errors.

For example

$$f(x) = \sum_{i=1}^{n} x_i^2 \text{ at } x_i = i \text{ for } i = 1 \ldots n$$

Then for $e_1 \in \mathbb{R}^n$

$$\frac{f(x + he_1) - f(x)}{h} = \frac{\partial f(x)}{\partial x_1} + h = 2x_1 + h = 2 + h$$

Floating Points

Given that the quantity needs floating point number representation in machine accuracy of 64 bits

$$\text{Roundoff error } = f(x + he_1)\,\epsilon \approx n^3 \frac{\epsilon}{3} \text{ with } \epsilon = 2^{-54} \approx 10^{-16}$$

Floating Points

Given that the quantity needs floating point number representation in machine accuracy of 64 bits

$$\text{Roundoff error } = f\left(x + he_1\right)\epsilon \approx n^3\frac{\epsilon}{3} \text{ with } \epsilon = 2^{-54} \approx 10^{-16}$$

For $h = \sqrt{\epsilon}$,as often is recommended

- The difference quotient has a rounding error of size

$$\frac{1}{3}n^3\sqrt{\epsilon} \approx \frac{1}{3}n^3 10^{-8}$$

Cinvestav

Now, Imagine $n = 1000$

Then Rounding Error

$$\frac{1}{3}1000^3\sqrt{\epsilon} \approx \frac{1}{3}1000000000 \times 10^{-8} = \frac{1}{3}100 \approx 33.333...$$

Now, Imagine $n = 1000$

Then Rounding Error

$$\frac{1}{3}1000^3\sqrt{\epsilon} \approx \frac{1}{3}1000000000 \times 10^{-8} = \frac{1}{3}100 \approx 33.333...$$

Ouch

- We cannot even get the sign correctly!!!

$$\frac{f(x + he_1) - f(x)}{h}$$

Cinvestav

In contrast Automatic Differentiation

It yields

- $2x_i$ in both its forward and reverse modes

In contrast Automatic Differentiation

It yields

- $2x_i$ in both its forward and reverse modes

You could assume that the derivatives are generated symbolically

- Actually is true in some sense, but $2x_i$ will be never be generated by Symbolic Differentiation

In contrast Automatic Differentiation

It yields

- $2x_i$ in both its forward and reverse modes

You could assume that the derivatives are generated symbolically

- Actually is true in some sense, but $2x_i$ will be never be generated by Symbolic Differentiation

In Symbolic Differentiation

- The numerical value of x_i is multiplied by 2 then returned as the gradient value.

Cinvestav

Outline

Example using Forward Differentiation

We will see the forward procedure later on

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} x_i^2 \text{ with } x_i = i \text{ for } i = 1, ..., n$$

Example using Forward Differentiation

We will see the forward procedure later on

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} x_i^2 \text{ with } x_i = i \text{ for } i = 1, ..., n$$

AD Initializes (Do not worry we will see this in more detail)

$$\begin{aligned} v_{i-n} =& i \text{ for } i = 1, ..., n \text{ (The input)} \\ \frac{\partial v_{i-n}}{\partial v_{j-n}} =& 0, \text{ but } i \neq j \ \frac{\partial v_{i-n}}{\partial v_{i-n}} = \dot{v}_{1-n} = 1 \end{aligned}$$

Cinvestav

Then, we have that

Apply the compositions

ϕ Functions	Derivatives
$v_1 = 1^2$	$\dot{v}_1 = \frac{\partial v_1}{\partial v_{1-n}} \dot{v}_{1-n} = 2 \times (1) \times 1 = 2$
$\vdots$	$\vdots$
$v_n = n^2$	0

Cinvestav

Then, we have that

Apply the compositions

ϕ Functions	Derivatives
$v_1 = 1^2$	$\dot{v}_1 = \frac{\partial v_1}{\partial v_{1-n}} \dot{v}_{1-n} = 2 \times (1) \times 1 = 2$
$\vdots$	$\vdots$
$v_n = n^2$	0

Therefore, we have at the end

$$\frac{\partial f}{\partial x_1}(x) = (2, 0, ..., 0)$$

Quite different from

Using a numerical difference, we have

$$\frac{f\left(\boldsymbol{x}+\boldsymbol{e}_1 h\right)-f\left(\boldsymbol{x}\right)}{h}-2<0$$

Quite different from

Using a numerical difference, we have

$$\frac{f\left(\boldsymbol{x}+\boldsymbol{e}_1 h\right)-f\left(\boldsymbol{x}\right)}{h}-2<0$$

Then for $n=10^j$ and $h=10^{-k}$

$$10^k\left[(h+1)^2-1\right]<2$$

Quite different from

Using a numerical difference, we have

$$\frac{f\left(\boldsymbol{x}+\boldsymbol{e}_1 h\right)-f\left(\boldsymbol{x}\right)}{h}-2<0$$

Then for $n=10^j$ and $h=10^{-k}$

$$10^k\left[(h+1)^2-1\right]<2$$

Finally, we have

$$k>-\log_{10} 3$$

Cinvestav

Therefore

It is possible to get into underflow

- by getting a $k > -\log_{10} 3$

Therefore

It is possible to get into underflow

- by getting a $k > -\log_{10} 3$

Therefore, we have that

- Automatic Differentiation allows to obtain the correct answer!!!

Outline

Cinvestav

For example

You have the following equation

$$f(x) = \prod_{i=1}^{n} x_i$$

For example

You have the following equation

$$f(x) = \prod_{i=1}^{n} x_i$$

Then, the gradient

$$\begin{aligned}
\nabla f(x) &= \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n}\right) = \left(\prod_{j \neq i} x_j\right)_{i=1\ldots n} \\
&= (x_2 \times x_3 \times \ldots \times x_i \times x_{i+1} \times \ldots \times x_{n-1} \times x_n, \\
&\quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
&\quad x_1 \times x_2 \times \ldots \times x_{i-1} \times x_{i+1} \times \ldots \times x_{n-1} \times x_n, \\
&\quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
&\quad x_1 \times x_2 \times \ldots \times x_{i-1} \times x_i \times \ldots \times x_{n-2} \times x_{n-1},)
\end{aligned}$$

Actually

Symbolic Differentiation will consume a lot of memory

- Instead AD will reuse the common expressions to improve performance and memory.

Actually

Symbolic Differentiation will consume a lot of memory

- Instead AD will reuse the common expressions to improve performance and memory.

However, Symbolic and Automatic Differentiation

- They make use of the chain rule to achieve their results

Automatic Differentiation Makes use of the Chain Rule

We had for $f(x(t), y(t))$

$$\frac{\partial f(x(t), y(t))}{\partial t} = \frac{\partial f(x(t), y(t))}{\partial x(t)} \cdot \frac{\partial x(t)}{\partial t} + \frac{\partial f(x(t), y(t))}{\partial y(t)} \cdot \frac{\partial y(t)}{\partial t}$$

Outline

Cinvestav

The User Insight

Difference quotients may sometimes be useful too

$$\frac{f\left(x+he_1\right)-f\left(x\right)}{h}$$

The User Insight

Difference quotients may sometimes be useful too

$$\frac{f\left(x+he_{1}\right)-f\left(x\right)}{h}$$

Computer Algebra packages

- They have really neat ways to simplify expressions.

The User Insight

Difference quotients may sometimes be useful too

$$\frac{f\left(x+he_1\right)-f\left(x\right)}{h}$$

Computer Algebra packages

- They have really neat ways to simplify expressions.

In contrast, current AD packages assume that

- That the given program calculates the underlying function efficiently

There

AD can automatize the gradient generation

- The best results will be obtained when AD takes advantage
  - the user's insight into the structure underlying the program

Cinvestav

Outline

Cinvestav

RNN Example

When you look at the recurrent neural network Elman [6]

$$\boldsymbol{h}_t = \sigma_h \left(W_{sd} \boldsymbol{x}_t + U_{sh} \boldsymbol{h}_{t-1} + b_h \right)$$

$$\boldsymbol{y}_t = \sigma_y \left(V_{os} \boldsymbol{h}_t \right)$$

$$L = \frac{1}{2} \left(\boldsymbol{y}_t - \boldsymbol{z}_t \right)^2$$

RNN Example

When you look at the recurrent neural network Elman [6]

$$\boldsymbol{h}_t = \sigma_h \left(W_{sd} \boldsymbol{x}_t + U_{sh} \boldsymbol{h}_{t-1} + b_h \right)$$
$$\boldsymbol{y}_t = \sigma_y \left(V_{os} \boldsymbol{h}_t \right)$$
$$L = \frac{1}{2} \left(\boldsymbol{y}_t - \boldsymbol{z}_t \right)^2$$

Here if you do blind AD sooner or later you have

$$\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_{t-1}} \times \frac{\partial \boldsymbol{h}_{t-1}}{\partial \boldsymbol{h}_{t-2}} \times \frac{\partial \boldsymbol{h}_{t-2}}{\partial \boldsymbol{h}_{t-3}} \times ... \times \frac{\partial \boldsymbol{h}_{k+1}}{\partial \boldsymbol{h}_k}$$

- This is known as Back Propagation Through Time (BPTT)

RNN Example

When you look at the recurrent neural network Elman [6]

$$\boldsymbol{h}_t = \sigma_h \left(W_{sd} \boldsymbol{x}_t + U_{sh} \boldsymbol{h}_{t-1} + b_h \right)$$

$$\boldsymbol{y}_t = \sigma_y \left(V_{os} \boldsymbol{h}_t \right)$$

$$L = \frac{1}{2} \left(\boldsymbol{y}_t - \boldsymbol{z}_t \right)^2$$

Here if you do blind AD sooner or later you have

$$\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_{t-1}} \times \frac{\partial \boldsymbol{h}_{t-1}}{\partial \boldsymbol{h}_{t-2}} \times \frac{\partial \boldsymbol{h}_{t-2}}{\partial \boldsymbol{h}_{t-3}} \times ... \times \frac{\partial \boldsymbol{h}_{k+1}}{\partial \boldsymbol{h}_k}$$

- This is known as Back Propagation Through Time (BPTT)

This is a problem given

- The Vanishing Gradient or Exploding Gradient

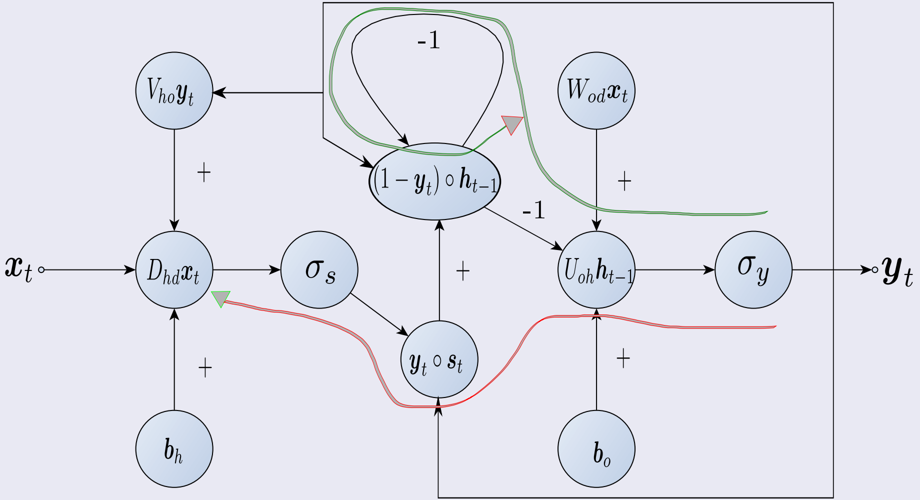
Here, you can modify the architecture

Using an intermediate layer using the Hadamard product $\circ$ we have

$$
\begin{aligned}
L &= \frac{1}{2}\left(\boldsymbol{y}_t - \boldsymbol{z}_t\right)^2 \\
\boldsymbol{y}_t &= \sigma_y\left(W_{od}\boldsymbol{x}_t + U_{oh}\boldsymbol{h}_{t-1} + \boldsymbol{b}_o\right) \\
\boldsymbol{s}_t &= \sigma_s\left(V_{ho}\boldsymbol{y}_t + D_{hd}\boldsymbol{x}_t + \boldsymbol{b}_h\right) \\
\boldsymbol{h}_t &= \left(1 - \boldsymbol{y}_t\right) \circ \boldsymbol{h}_{t-1} + \boldsymbol{y}_t \circ \boldsymbol{s}_t
\end{aligned}
$$

Therefore

You have multiple paths of derivatives



One of them

It can be seen

- That one of the paths can take you to BPTT

The Other One

The other gets you into a more Markovian Property

- This allows to to get a Backpropagation that does not require the BPTT

The Other One

The other gets you into a more Markovian Property

- This allows to to get a Backpropagation that does not require the BPTT

The Other One

The other gets you into a more Markovian Property

- This allows to to get a Backpropagation that does not require the BPTT

How? For example, the derivative of L with respect to D_{hd}

$$\frac{\partial L}{\partial D_{hd}} = \frac{\partial L}{\partial \boldsymbol{y}_t} \times \frac{\partial \boldsymbol{y}_t}{\partial net_y} \times \frac{\partial net_y}{\partial \boldsymbol{h}_{t-1}} \times \frac{\partial \boldsymbol{h}_{t-1}}{\partial \boldsymbol{s}_{t-2}} \times \frac{\partial \boldsymbol{s}_{t-2}}{net_s} \times \frac{net_s}{\partial D_{hd}}$$

Therefore

You do not have

- The Backpropagation through time... By assuming a Markovian Property...
  - Or its big brother truncated backpropagation

Therefore

You do not have

- The Backpropagation through time... By assuming a Markovian Property...
  - Or its big brother truncated backpropagation

Because Backpropagation Through Time

- Makes the process of obtaining the gradients unstable...

Thus

A great simplifying step

- Here resound trues the phrase
  - “AD taking advantage of the user’s insight”

Outline

Cinvestav

A Simple Example

Here, we have the following ideas

- Some of the floating point values, generated by the AD, will be stored in variables of the program,

A Simple Example

Here, we have the following ideas

- Some of the floating point values, generated by the AD, will be stored in variables of the program,
- Other operations will be held until overwritten or discarded.

Cinvestav

A Simple Example

Here, we have the following ideas

- Some of the floating point values, generated by the AD, will be stored in variables of the program,
- Other operations will be held until overwritten or discarded.

Thus, we will introduce the concept

- **Evaluation Trace** which is basically a record of a particular run of a given program.

A Simple Example

Here, we have the following ideas

- Some of the floating point values, generated by the AD, will be stored in variables of the program,
- Other operations will be held until overwritten or discarded.

Thus, we will introduce the concept

- **Evaluation Trace** which is basically a record of a particular run of a given program.

This Evaluation Trace stores

- Input variables,

A Simple Example

Here, we have the following ideas

- Some of the floating point values, generated by the AD, will be stored in variables of the program,
- Other operations will be held until overwritten or discarded.

Thus, we will introduce the concept

- **Evaluation Trace** which is basically a record of a particular run of a given program.

This Evaluation Trace stores

- Input variables,
- Sequence of floating point generated by the CPU

Cinvestav

A Simple Example

Here, we have the following ideas

- Some of the floating point values, generated by the AD, will be stored in variables of the program,
- Other operations will be held until overwritten or discarded.

Thus, we will introduce the concept

- **Evaluation Trace** which is basically a record of a particular run of a given program.

This Evaluation Trace stores

- Input variables,
- Sequence of floating point generated by the CPU
- Operations that are used for it

Example

A simple example

$$y = f\left(x_1, x_2\right) = \left[\sin\left(\frac{x_1}{x_2}\right) + \frac{x_1}{x_2} - \exp\left(x_2\right)\right] \times \left[\frac{x_1}{x_2} - \exp\left(x_2\right)\right]$$

Example

A simple example

$$y = f\left(x_1, x_2\right) = \left[\sin\left(\frac{x_1}{x_2}\right) + \frac{x_1}{x_2} - \exp\left(x_2\right)\right] \times \left[\frac{x_1}{x_2} - \exp\left(x_2\right)\right]$$

We wish to calculate $y = f\left(x_1, x_2\right)$

- With $x_1 = 1.5,\ x_2 = 0.5$

Evaluation Trace/Forward Procedure

We have the table for the evaluation of the function

$$v_{-1} = x_1 = 1.5$$
$$v_0 = x_2 = 0.5$$

$$v_1 = \frac{v_{-1}}{v_0} = \frac{1.5}{0.5} = 3.0$$
$$v_2 = \sin(v_1) = \sin(3.0) = 0.1411$$
$$v_3 = \exp(v_0) = \exp(0.5) = 1.6487$$
$$v_4 = v_1 - v_3 = 3.0 - 1.6487 = 1.3513$$
$$v_5 = v_2 + v_4 = 0.1411 + 1.3413 = 1.4924$$
$$v_6 = v_5 \times v_4 = 1.4924 \times 1.3513 = 2.0167$$

$$y = v_6 = 2.0167$$

Evaluation Trace/Forward Procedure

Input Variables

$$\Rightarrow \boldsymbol{v_{-1} = x_1 = 1.5}$$
$$\Rightarrow \boldsymbol{v_0 = x_2 = 0.5}$$

$$v_1 = \frac{v_{-1}}{v_0} = \frac{1.5}{0.5} = 3.0$$
$$v_2 = \sin(v_1) = \sin(3.0) = 0.1411$$
$$v_3 = \exp(v_0) = \exp(0.5) = 1.6487$$
$$v_4 = v_1 - v_3 = 3.0 - 1.6487 = 1.3513$$
$$v_5 = v_2 + v_4 = 0.1411 + 1.3413 = 1.4924$$
$$v_6 = v_5 \times v_4 = 1.4924 \times 1.3513 = 2.0167$$

$$y = v_6 = 2.0167$$

Cinvestav

Evaluation Trace/Forward Procedure

Evaluation Functions

$$v_{-1} = x_1 = 1.5$$
$$v_0 = x_2 = 0.5$$

$$\Rightarrow v_1 = \frac{v_{-1}}{v_0} = \frac{1.5}{0.5} = 3.0$$
$$\Rightarrow v_2 = \sin(v_1) = \sin(3.0) = 0.1411$$
$$\Rightarrow v_3 = \exp(v_0) = \exp(0.5) = 1.6487$$
$$\Rightarrow v_4 = v_1 - v_3 = 3.0 - 1.6487 = 1.3513$$
$$\Rightarrow v_5 = v_2 + v_4 = 0.1411 + 1.3413 = 1.4924$$
$$\Rightarrow v_6 = v_5 \times v_4 = 1.4924 \times 1.3513 = 2.0167$$

$$y = v_6 = 2.0167$$

A Cautionary Note

Normally

- Programmers will try to rearrange this execution trace to improve performance through parallelism.
  - After all we want to use all the cores...

A Cautionary Note

Normally

- Programmers will try to rearrange this execution trace to improve performance through parallelism.
  - After all we want to use all the cores...

Thus

- Subexpressions will be algorithmically exploited by the AD to improve performance.

A Cautionary Note

Normally

- Programmers will try to rearrange this execution trace to improve performance through parallelism.
  - After all we want to use all the cores...

Thus

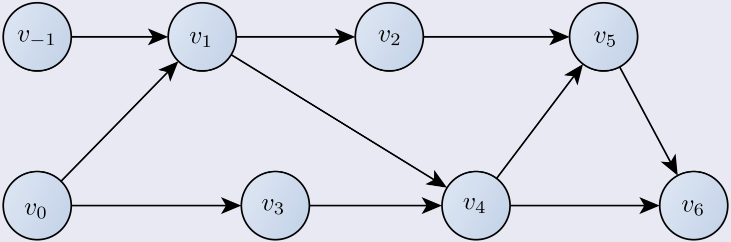
- Subexpressions will be algorithmically exploited by the AD to improve performance.

It is usually more convenient to use

- The so called "computational graph"

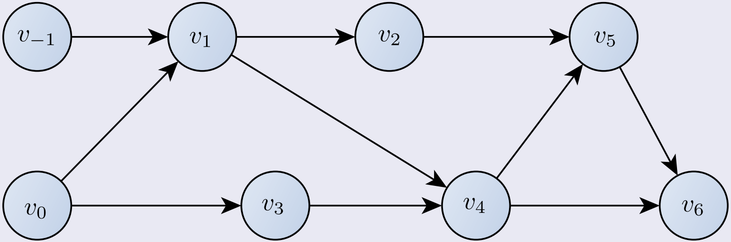
Computational Graph

A Simpler Version



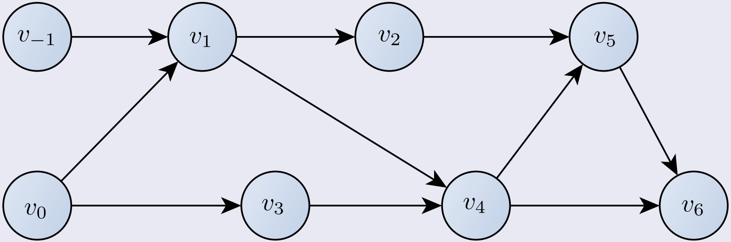
Computational Graph

A Simpler Version



Computational Graph

A Simpler Version



Please take a look at section in **Chapter 2 A Framework for Evaluating Functions**

- At the book [5]
  - Andreas Griewank and Andrea Walther, **Evaluating derivatives: principles and techniques of algorithmic differentiation** vol. 105, (Siam, 2008).

Cinvestav

Outline

Cinvestav

What can be evaluated?

We want to differentiate a more or less arbitrary vector-valued

$$\text{function } F : \mathcal{D} \subset \mathbb{R}^n \rightarrow \mathbb{R}^m$$

What can be evaluated?

We want to differentiate a more or less arbitrary vector-valued

function $F : \mathcal{D} \subset \mathbb{R}^n \to \mathbb{R}^m$

Actually, we want to know the existence of well defined matrix function

Jacobian $F' : \mathcal{D} \subset \mathbb{R}^n \to \mathbb{R}^{m \times n}$

A Little Bit of Notation

In general, we assume quantities v_i such

$$\underbrace{v_{1-n}, ..., v_0}_{x} v_1, ..., v_{l-m-1} \underbrace{v_{l-m+1}, ..., v_l}_{y}$$

Then, we have

1. $v_{1-n}, ..., v_0$ are the initial input variables
2. $v_{l-m+1}, ..., v_l$ the output variables
3. $v_1, ..., v_{l-m-1}$ the intermediate functions

Cinvestav

Additionally

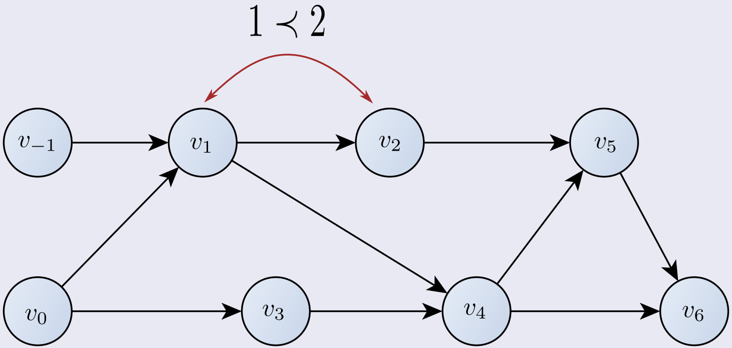
Where each value v_i with $i > 0$ is obtained by applying an elemental function ϕ

$$v_i = \phi_i \left(v_j\right)_{j \prec i}$$

- notation $j \prec i$ means v_i depends directly on v_j

Remember the Computational Graph

For example $1 \prec 2$ and $0 \prec 1$, v_2 depends directly on v_1 and also v_0



At the Computational Graph

The Acyclic Graph

- These data dependence relations can be visualized as an acyclic graph

At the Computational Graph

The Acyclic Graph

- These data dependence relations can be visualized as an acyclic graph

The Vertices

- The set of vertices are simply the variables v_i for $i = 1 - n ... l$,

At the Computational Graph

The Acyclic Graph

- These data dependence relations can be visualized as an acyclic graph

The Vertices

- The set of vertices are simply the variables v_i for $i = 1 - n...l$,

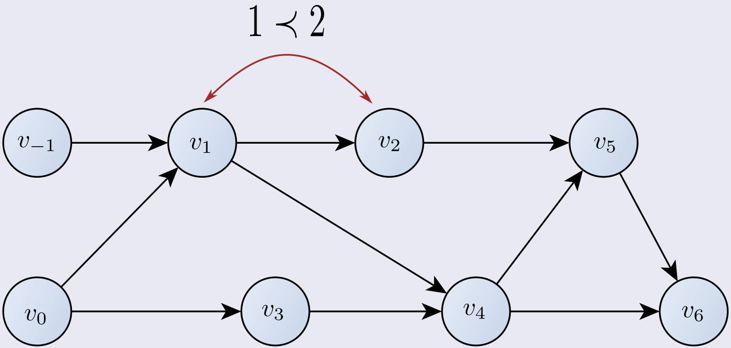
The Arcs

- An arc runs from v_j to v_i exactly when $j \prec i$.

Not only that

The roots of the graph represent the independent variables

- $x_j = v_{j-n}$ for $j = 1...n$,



Then, for the application of the chain rule

It is useful to associate with each elemental function ϕ_i the state transformation

$$\mathsf{v}_i = \Phi_i\left(\mathsf{v}_{i-1}\right) \text{ with } \Phi_i : \mathbb{R}^{n+l} \to \mathbb{R}^{n+l}$$

Then, for the application of the chain rule

It is useful to associate with each elemental function ϕ_i the state transformation

$$\mathsf{v}_i = \Phi_i\left(\mathsf{v}_{i-1}\right) \text{ with } \Phi_i : \mathbb{R}^{n+l} \to \mathbb{R}^{n+l}$$

where v_i is a vector of a certain form

$$\mathsf{v}_i = \left(v_{1-n}, ..., v_i, 0, ..., 0\right)^T$$

Then, for the application of the chain rule

It is useful to associate with each elemental function ϕ_i the state transformation

$$\mathsf{v}_i = \Phi_i\left(\mathsf{v}_{i-1}\right) \text{ with } \Phi_i : \mathbb{R}^{n+l} \to \mathbb{R}^{n+l}$$

where v_i is a vector of a certain form

$$\mathsf{v}_i = \left(v_{1-n}, ..., v_i, 0, ..., 0\right)^T$$

In other words

- Φ_i sets of v_i to $\phi_i\left(v_j\right)_{j \prec i}$ and keeps all other components v_j for $j \neq i$ unchanged.

Cinvestav

Outline

We have a general procedure

General Evaluation Procedure

$v_{i-n} = x_i$	$i = 1...n$	independent variables
$v_i = \varphi\left(v_j\right)_{j \prec i}$	$i = 1...n$	The use of function to produce new variables
$y_{m-i} = v_{l-i}$	$i = 1...m-1$	dependent variables

Thus, we have that

We can encapsulate it a nonlinear system of equations

$$0 = E\left(x; v\right) \equiv \left(\varphi_i\left(u_i\right) - v_i\right)_{i=1-n,\ldots,l}$$

Thus, we have that

We can encapsulate it a nonlinear system of equations

$$0 = E\left(x; v\right) \equiv \left(\varphi_i\left(u_i\right) - v_i\right)_{i=1-n,...,l}$$

where the first n components of E are defined as the initialization functions

$$\varphi_i\left(u_i\right) = x_{i+n} \text{ for } i = 1-n, ..., 0$$

Thus, we have that

We can encapsulate it a nonlinear system of equations

$$0 = E(x; v) \equiv (\varphi_i(u_i) - v_i)_{i=1-n,\ldots,l}$$

where the first n components of E are defined as the initialization functions

$$\varphi_i(u_i) = x_{i+n} \text{ for } i = 1-n, \ldots, 0$$

We may assume without loss of generality

- The dependent variables are mutually independent.

$$y_{m-i} = v_{l-i} \text{ for } 0 \leq i \leq n$$

Some definitions

We define c_{ij}

$$c_{ij} = c_{ij}\left(u_i\right) = \frac{\partial \varphi_i}{\partial v_j} \text{ for } 1-n \leq i,j \leq l$$

In this way

We have that $i < 1$ or $j > l - m$ implies

$$c_{ij} \equiv 0$$

In this way

We have that $i < 1$ or $j > l - m$ implies

$$c_{ij} \equiv 0$$

These derivatives will be called elemental partials throughout

- The Jacobian of E with respect to the $n + l$ variables v_j for $j = 1 - n \ldots l$ is a unitary lower triangular matrix

$$E'(x; v) = (c_{ij} - \delta_{ij})_{j=1-n,..,l}^{i=1-n,\ldots,l} = C - I$$

- Kronecker Delta $\delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$

Or as they say

$C - I$

$$
C-I=\begin{pmatrix}
-1 & 0 & \cdots & 0 & 0 & \cdots & \cdots & 0 & 0 & \cdots & \cdots & 0\\
0 & \ddots & \ddots & \vdots & \vdots & & & & \vdots & & & \vdots\\
\vdots & \ddots & \ddots & 0 & \vdots & & & & \vdots & & & \vdots\\
0 & \cdots & 0 & -1 & 0 & \cdots & \cdots & 0 & 0 & \cdots & \cdots & 0\\
\times & \cdots & \cdots & \times & -1 & 0 & \cdots & 0 & 0 & \cdots & \cdots & 0\\
 & & & & \times & \ddots & \ddots & & & \vdots & & \vdots\\
 & & & & & \ddots & \ddots & \ddots & & \vdots & & \vdots\\
\times & \cdots & \cdots & \times & \times & \cdots & \times & -1 & 0 & \cdots & \cdots & 0\\
\times & \cdots & \cdots & \times & \times & \cdots & \cdots & \times & -1 & & 0 & 0\\
\vdots & & & \vdots & \vdots & & & \vdots & 0 & & \ddots & \vdots\\
\vdots & & & \vdots & \vdots & & & \vdots & \vdots & \ddots & \ddots & 0\\
\times & \cdots & \cdots & \times & \times & \cdots & \cdots & \times & 0 & & 0 & -1
\end{pmatrix}
=\begin{pmatrix}
\underbrace{\begin{matrix}-I\\ B\\ R\end{matrix}}_{n} & \begin{matrix}0\\ L-I\\ T\end{matrix} & \underbrace{\begin{matrix}0\\ 0\\ -I\end{matrix}}_{m}
\end{pmatrix}\Bigg\}l
$$

First, we noticed something simple

It is a unitary matrix

- All element in the diagonal different from zero $\Rightarrow$ the matrix is inveretible

Cinvestav

First, we noticed something simple

It is a unitary matrix

- All element in the diagonal different from zero $\Rightarrow$ the matrix is inveretible

Therefore

- $-E'(x;v) = I - C$ can never be singular

Cinvestav

Then, we have that

The Implicit Function Theorem

- Let $F : \mathbb{R}^{n+m} \to \mathbb{R}^m$ be a continuously differentiable function, and a point $\left(x_1^0, x_2^0, ..., x_{m+n}^0\right)$ so $F\left(x_1^0, x_2^0, ..., x_{m+n}^0\right) = c$. If $\frac{\partial F\left(x_1^0, x_2^0, ..., x_{m+n}^0\right)}{\partial x_{m+n}} \neq 0$, then there exist a neighborhood of $\left(x_1^0, x_2^0, ..., x_{m+n}^0\right)$ so whatever $\left(x_1, ..., x_{n+m-1}\right)$ is close enough to $\left(x_1^0, ..., x_{m+n-1}^0\right)$, there is a unique z so that $F\left(x_1, ..., x_{n+m-1}, z\right) = c$. Furthermore, $z = g\left(x_1, ..., x_{n+m-1}\right)$ a continuous function of $\left(x_1, ..., x_{n+m-1}\right)$.

Then, we have that

The Implicit Function Theorem

- Let $F : \mathbb{R}^{n+m} \to \mathbb{R}^m$ be a continuously differentiable function, and a point $\left(x_1^0, x_2^0, ..., x_{m+n}^0\right)$ so $F\left(x_1^0, x_2^0, ..., x_{m+n}^0\right) = c$. If $\frac{\partial F\left(x_1^0, x_2^0, ..., x_{m+n}^0\right)}{\partial x_{m+n}} \neq 0$, then there exist a neighborhood of $\left(x_1^0, x_2^0, ..., x_{m+n}^0\right)$ so whatever $\left(x_1, ..., x_{n+m-1}\right)$ is close enough to $\left(x_1^0, ..., x_{m+n-1}^0\right)$, there is a unique z so that $F\left(x_1, ..., x_{n+m-1}, z\right) = c$. Furthermore, $z = g\left(x_1, ..., x_{n+m-1}\right)$ a continuous function of $\left(x_1, ..., x_{n+m-1}\right)$.

Then if we have that given $E(x; v)$ =0

- Uniquely defines all $v_i's$ in particular the ones defined as $y = F\left(x\right)$

Actually

$E'(x; v) = C - I$ allow to obtain

- A general "elimination method" to compute a compact Jacobian $F'(x)$ as Schur complement

Outline

Cinvestav

Example of the Forward Mode

Suppose we want to differentiate $y = f(x_1, x_2)$ with respect to x_1

- We consider x_1 as an independent variable and y as a dependent variable.

Example of the Forward Mode

Suppose we want to differentiate $y = f(x_1, x_2)$ with respect to x_1

- We consider x_1 as an independent variable and y as a dependent variable.

We can work the numerical value of the $y = f(x_1, x_2)$

- By getting the numerical derivative of each of its components

Example of the Forward Mode

Suppose we want to differentiate $y = f(x_1, x_2)$ with respect to x_1

- We consider x_1 as an independent variable and y as a dependent variable.

We can work the numerical value of the $y = f(x_1, x_2)$

- By getting the numerical derivative of each of its components

Something like

$$\dot{v}_i = \frac{\partial v_i}{\partial x_1}$$

Therefore, we get

We have the Procedure

$v_{-1} = x_1 = 1.5$ $v_0 = x_2 = 0.5$	$\dot{v}_{-1} = 1.0$ $\dot{v}_1 = 0.0$
$v_1 = \frac{v_{-1}}{v_0} = \frac{1.5}{0.5} = 3.0$ $v_2 = \sin(v_1) = \sin(3.0) = 0.1411$ $v_3 = \exp(v_0) = \exp(0.5) = 1.6487$ $v_4 = v_1 - v_3 = 3.0 - 1.6487 = 1.3513$ $v_5 = v_2 + v_4 = 0.1411 + 1.3413 = 1.4924$ $v_6 = v_5 \times v_4 = 1.4924 \times 1.3513 = 2.0167$	$\dot{v}_1 = \frac{\partial v_1}{\partial v_{-1}} \dot{v}_{-1} + \frac{\partial v_1}{\partial v_0} \dot{v}_0 = 2.0$ $\dot{v}_2 = \cos(v_1) \dot{v}_1 = -1.98$ $\dot{v}_3 = v_3 \dot{\times} v_1 = 0.0$ $\dot{v}_4 = \dot{v}_1 - \dot{v}_3 = 2.0$ $\dot{v}_5 = \dot{v}_2 + \dot{v}_4 = 0.02$ $\dot{v}_6 = \dot{v}_5 \times v_4 + v_5 \times \dot{v}_4 = 3.0118$
$y = v_6 = 2.0167$	$\dot{y} = 3.0118$

The first Column of this process

It can be seen as an automatic procedure

v_{i-n}	$i = 1 \ldots n$
$v_i = \varphi_i \left(v_j\right)_{j \prec i}$	$i = 1 \ldots l$
$y_{m-i} = v_{l-i}$	$i = m-1 \ldots 0$

In a similar way

We can obtain $\frac{\partial f(x_1,x_2)}{\partial x_2}$

- However, it can be more efficient to redefine the $\dot{v}_i$ as vectors for efficiency!!!

Outline

Cinvestav

Forward propagation of Tangents

Remarks

- As you can see the second column of the evaluation procedure is done in a mechanical way

Forward propagation of Tangents

Remarks

- As you can see the second column of the evaluation procedure is done in a mechanical way

This increase the size

- Basically, twice the size of the original simple evaluation.

We have the following

We have the chain rule

$$\dot{y}(t) = \frac{\partial F(x(t))}{\partial t} = F'(x(t))\,\dot{x}(t)$$

We have the following

We have the chain rule

$$\dot{y}(t) = \frac{\partial F(x(t))}{\partial t} = F'(x(t))\,\dot{x}(t)$$

Where

- $F'(x) \in \mathbb{R}^{m \times n}$ is the Jacobian Matrix

We have the following

We have the chain rule

$$\dot{y}(t) = \frac{\partial F(x(t))}{\partial t} = F'(x(t))\,\dot{x}(t)$$

Where

- $F'(x) \in \mathbb{R}^{m \times n}$ is the Jacobian Matrix

Here, we will be tempted to calculate $\dot{y}(t)$

- By evaluating the full Jacobian $F'(x)$ then multiplying by $\dot{x}(t)$

Cinvestav

However

Such approach is quite uneconomically

- Unless many tangents need to be calculated as in the Newton Step.

However

Such approach is quite uneconomically

- Unless many tangents need to be calculated as in the Newton Step.

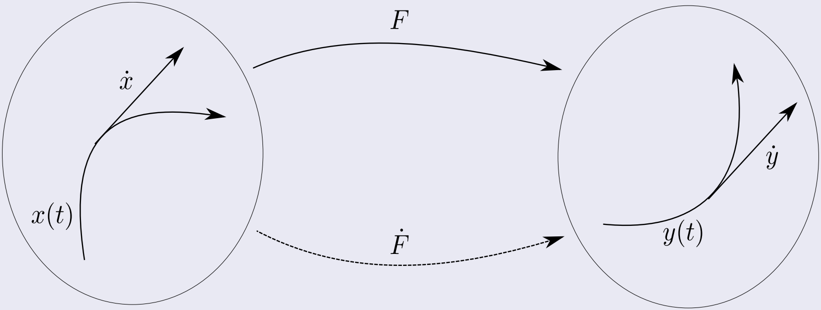
A simpler version, differentiate the first column of the table

$v_{i-n} = x_i$	$i = 1, ..., n$
$v_i = \phi_i \left(v_j\right)_{j \prec i}$	$i = 1, ..., l$
$y_{m-i} = v_{l-i}$	$i = m-1, ..., 0$

- $j \prec i$ v_i depends directly v_j (The graph propagation of the dependencies)

Which can be seen as Forward Propagation of Tangents

Basically, we can think of the forward mode as a propagation of tangents



$$\dot{y}\left(t\right)=\frac{\partial}{\partial t}F\left(x\left(t\right)\right)=F'\left(x\left(t\right)\right)\dot{x}\left(t\right)$$

Cinvestav

The Automatic Procedure

Therefore, we have the following automatic procedure

- $j \prec i$ v_i **depends directly on** v_j **and** $u_i = \left(v_j\right)_{j \prec i} \in \mathbb{R}^{n_i}$

$v_{i-n} \equiv x_i$ $\dot{v}_{i-n} \equiv \dot{x}_i$	$i = 1 \ldots n$
$v_i \equiv \phi_i \left(v_j\right)_{j \prec i}$ $i = 1 \ldots l$ $\dot{v}_i \equiv \sum_{j \prec i} \frac{\partial \phi_i(u_j)}{\partial v_j} \dot{v}_j$	$i = 1 \ldots l$
$y_{m-i} \equiv v_{l-i}$ $\dot{y}_{m-i} \equiv \dot{v}_{l-i}$	$i = m-1 \ldots 0$

Cinvestav

Therefore

Each element assignment $v_i = \phi_i\left(u_i\right)$

- You have the corresponding

$$\dot{v}_i = \sum_{j \prec i} \frac{\partial \phi_i\left(u_j\right)}{\partial v_j} \times \dot{v}_j = \sum_{j \prec i} c_{ij} \times \dot{v}_j$$

Therefore

Each element assignment $v_i = \phi_i(u_i)$

- You have the corresponding

$$\dot{v}_i = \sum_{j \prec i} \frac{\partial \phi_i(u_j)}{\partial v_j} \times \dot{v}_j = \sum_{j \prec i} c_{ij} \times \dot{v}_j$$

Abbreviating $\dot{u}_i = (\dot{v}_j)_{j \prec i}$

$$\dot{v}_i = \dot{\phi}_i(u_i, \dot{u}_i) = \phi_i'(u_i)\, \dot{u}_i$$

Cinvestav

Therefore

Each element assignment $v_i = \phi_i(u_i)$

- You have the corresponding

$$\dot{v}_i = \sum_{j \prec i} \frac{\partial \phi_i(u_j)}{\partial v_j} \times \dot{v}_j = \sum_{j \prec i} c_{ij} \times \dot{v}_j$$

Abbreviating $\dot{u}_i = (\dot{v}_j)_{j \prec i}$

$$\dot{v}_i = \dot{\phi}_i(u_i, \dot{u}_i) = \phi_i'(u_i)\,\dot{u}_i$$

Where $\dot{\phi}_i = \mathbb{R}^{2n_i} \to \mathbb{R}$

- It is called the tangent function associated with the elemental ϕ_i.

Now

Question

- What is the correct order of evaluation?

Why the question?

Until now, we have always placed the tangent statement yielding $\dot{v}_i$ after the underlying value v_i

- This order of calculation seems natural and certainly yields correct results as long as there is no overwriting.

Why the question?

Until now, we have always placed the tangent statement yielding $\dot{v}_i$ after the underlying value v_i

- This order of calculation seems natural and certainly yields correct results as long as there is no overwriting.

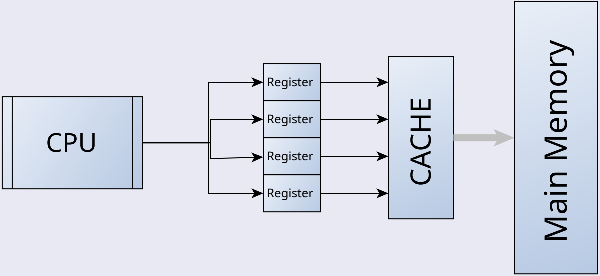
Then the order of $2l$ statements in the middle part of Table does not matter

$v_{i-n} \equiv x_i$ $\dot{v}_{i-n} \equiv \dot{x}_i$	$i = 1...n$
$v_i \equiv \phi_i\left(v_j\right)_{j \prec i}\ i = 1...l$ $\dot{v}_i \equiv \sum_{j \prec i} \frac{\partial \phi_i(u_j)}{\partial v_j} \dot{v}_j$	$i = 1...l$
$y_{m-i} \equiv v_{l-i}$ $\dot{y}_{m-i} \equiv \dot{v}_{l-i}$	$i = m - 1...0$

Here, we have a big problem in Cache

Imagine that we have a single block of memory to hold

- For v_i and its arguments v_j live in the same memory cell on the cache memory



This is known as Cache Aliasing

Definition

- Cache aliasing occurs when multiple mappings to a physical page of memory have conflicting caching states, such as cached and uncached.
  - the same physical address can be mapped to multiple virtual addresses.

This is known as Cache Aliasing

Definition

- Cache aliasing occurs when multiple mappings to a physical page of memory have conflicting caching states, such as cached and uncached.
  - the same physical address can be mapped to multiple virtual addresses.

On ARMv4 and ARMv5 processors, cache is organized as a virtual-indexed, virtual-tagged (VIVT)

- Cache lookups are faster because the translation look-aside buffer (TLB) is not involved in matching cache lines for a virtual address.

This is known as Cache Aliasing

Definition

- Cache aliasing occurs when multiple mappings to a physical page of memory have conflicting caching states, such as cached and uncached.
  - the same physical address can be mapped to multiple virtual addresses.

On ARMv4 and ARMv5 processors, cache is organized as a virtual-indexed, virtual-tagged (VIVT)

- Cache lookups are faster because the translation look-aside buffer (TLB) is not involved in matching cache lines for a virtual address.

However

- This caching method does require more frequent cache flushing because of cache aliasing.

Then

The value of $\dot{v}_i = \dot{\phi}_i\left(u_i, \dot{u}_i\right)$ it will incorrect

- Once we update $v_i = \phi_i\left(u_i\right)$

Then

The value of $\dot{v}_i = \dot{\phi}_i\left(u_i, \dot{u}_i\right)$ it will incorrect

- Once we update $v_i = \phi_i\left(u_i\right)$

Asifor and Tapenade [7, 3]

- They put the derivative statement ahead of the original assignment and update before the erasing the original statement.

Then

The value of $\dot{v}_i = \dot{\phi}_i(u_i, \dot{u}_i)$ it will incorrect

- Once we update $v_i = \phi_i(u_i)$

Asifor and Tapenade [7, 3]

- They put the derivative statement ahead of the original assignment and update before the erasing the original statement.

On the other hand

- For most univariate functions $v = \phi(u)$ is better to obtain the undifferentiated value first
  - Then to use it into the tangent function $\dot{\phi}$

Cinvestav

In this way

We will list φ and $\dot{\varphi}$

Side by side in a common bracket to indicate that they should be evaluated simultaneously

In this way

We will list φ and $\dot{\varphi}$

Side by side in a common bracket to indicate that they should be evaluated simultaneously

Then

- sharing results is immediate.

Classic Tangent Operations

We have a series of improvements on the tangent equations

ϕ	$\left[\phi, \dot{\phi}\right]$
$v = c$	$v = c,\ \dot{v} = 0$
$v = v \pm w$	$v = v \pm w$ $\dot{v} = \dot{v} \pm \dot{w}$
$v = u \times w$	$\dot{v} = \dot{u} \times w + u \times \dot{w}$ $v = u \times w$
$v = {}^{1}\!/_{u}$	$v = {}^{1}\!/_{u}$ $\dot{v} = -v \times (v \times \dot{u})$

ϕ	$\left[\phi, \dot{\phi}\right]$
$v = u^c$	$v = \frac{\dot{u}}{u}; v = u^c$ $\dot{v} = v \times (v \times \dot{u})$
$v = \sqrt{u}$	$v = \sqrt{u}$ $v = 0.5 \times \frac{\dot{u}}{v}$
$v = \exp(u)$	$v = \exp(u)$ $\dot{v} = v * \dot{u}$
$v = \log(u)$	$\dot{v} = {}^{\dot{u}}\!/_{u}$ $v = \log(u)$
$v = \sin(u)$	$\dot{v} = \cos(u) \times \dot{u}$ $v = \sin(u)$

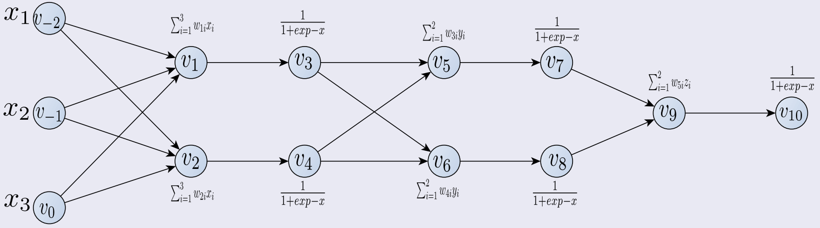
Cinvestav

Outline

Cinvestav

Now Imagine the following network

Something simple for our sake



Forward mode to get gradient of x_1

$v_{-14} = w_{11}, ..., v_{-6} = w_{16}, v_{-5} = w_{21}, ..., v_{-2} = x_1, v_{-1} = x_2, v_0 = x_3$
$\dot{v}_{-14} = 1, \dot{v}_{-10} = 0, ..., \dot{v_0} = 0$
$v_1 = \sum_{i=1}^{3} w_{1i}x_i \ , \dot{v}_1 = x_1$
$v_2 = \sum_{i=1}^{3} w_{2i}x_i \ , \dot{v}_2 = 0$
$v_3 = \frac{1}{1+\exp(-v_1)} \ , \dot{v}_3 = v_3 \left[1 - v_3\right] x_{11}$
$v_4 = \frac{1}{1+\exp(-v_2)} \ , \dot{v}_4 = 0$
$v_5 = \sum_{i=1}^{3} w_{3i}v_i, \ \dot{v}_5 = w_{31} \times \dot{v}_3$
$v_6 = \sum_{i=1}^{3} w_{4i}v_i, \ \dot{v}_6 = w_{41} \times \dot{v}_3$
$v_7 = \frac{1}{1+\exp(-v_5)}, \ \dot{v}_7 = v_7 \left[1 - v_7\right] \times \dot{v}_5$
$v_8 = \frac{1}{1+\exp(-v_6)}, \ \dot{v}_8 = v_8 \left[1 - v_8\right] \times \dot{v}_6$
$v_9 = \sum_{i=1}^{2} w_{5i}v_i, \ \dot{v}_9 = w_{51} \times \dot{v}_7 + w_{32} \times \dot{v}_8$
$v_{10} = \frac{1}{1+\exp(-v_9)}, \ \dot{v}_{10} = v_{10} \left[1 - v_{10}\right] \times \dot{v}_9$

Outline

Complexity of the Procedure

Time Complexity

$$TIME\left\{F\left(x\right), F'\left(x\right)\dot{x}\right\} \leq w_{tan} TIME\left\{F\left(x\right)\right\}$$

- Where $w_{tan} \in \left[2, \frac{5}{2}\right]$

Cinvestav

Complexity of the Procedure

Time Complexity

$$TIME\left\{F\left(x\right), F'\left(x\right)\dot{x}\right\} \leq w_{tan} TIME\left\{F\left(x\right)\right\}$$

- Where $w_{tan} \in \left[2, \frac{5}{2}\right]$

Space Complexity

$$SPACE\left\{F\left(x\right), F'\left(x\right)\dot{x}\right\} = 2SPACE\left\{F\left(x\right)\right\}$$

Outline

Cinvestav

Here, an essential observation

The cost of evaluating derivatives by propagating them forward

- it increases linearly with number of directions $\dot{x}$ along which we want to differentiate.

Here, an essential observation

The cost of evaluating derivatives by propagating them forward

- it increases linearly with number of directions $\dot{x}$ along which we want to differentiate.

It looks inevitable

- But it is possible to avoid these complexity by
  - Observing that the gradient of a single dependent variable could be obtained for a fixed multiple of the cost of evaluating the underlying scalar-valued function.

We choose instead an output variable

We use the term "reverse mode" for this technique

- Because the label "backward differentiation" is well established [8, 9].

We choose instead an output variable

We use the term "reverse mode" for this technique

- Because the label "backward differentiation" is well established [8, 9].

Therefore, for an output $f(x_1, x_2)$

- We have for each variable v_i

$$\overline{v}_i = \frac{\partial y}{\partial v_i} \text{ (Adjoint Variable)}$$

Cinvestav

Actually

This is an abuse of notation

- We mean a new independent variable δ_i

$$\overline{v}_i = \frac{\partial y}{\partial \delta_i} \text{ (Adjoint Variable)}$$

Actually

This is an abuse of notation

- We mean a new independent variable δ_i

$$\overline{v}_i = \frac{\partial y}{\partial \delta_i} \text{ (Adjoint Variable)}$$

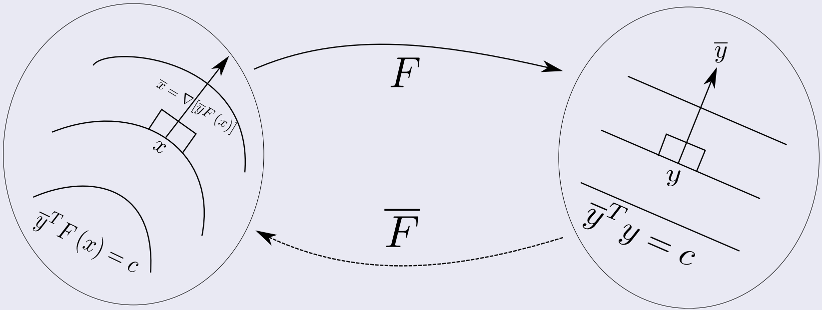
Which can be thought as adding a small numerical value δ_i to v_i

$$v_i + \delta_i \to f(x_1, x_2) + \overline{v}_i \delta_i$$

- As a perturbation in variational calculus

Actually, you propagate the Normal vectors

Actually, $\overline{y}$ and $\overline{v}_i$ are normals or cotangents



Then, we have

The following sought mapping

$$\overline{x} = \nabla \left[\overline{y}^T F(x) \right] = \overline{y}^T F'(x)$$

Then, we have

The following sought mapping

$$\overline{x} = \nabla \left[\overline{y}^T F(x) \right] = \overline{y}^T F'(x)$$

Observation

- Here, $\overline{y}$ is a fixed vector that plays a dual role to the domain direction $\dot{x}$.

Then, we have

The following sought mapping

$$\overline{x} = \nabla\left[\overline{y}^T F(x)\right] = \overline{y}^T F'(x)$$

Observation

- Here, $\overline{y}$ is a fixed vector that plays a dual role to the domain direction $\dot{x}$.

In the Forward Procedure, you compute

$$\dot{y} = F'(x)\,\dot{x} = \dot{F}(x, \dot{x})$$

Cinvestav

Instead

In the Reverse Procedure, you compute

$$\overline{x}^T = \overline{y}^T F'(x) \equiv \overline{F}(x, \overline{y})$$

Instead

In the Reverse Procedure, you compute

$$\overline{x}^T = \overline{y}^T F'(x) \equiv \overline{F}(x, \overline{y})$$

Where F and $\overline{F}$ are evaluated together

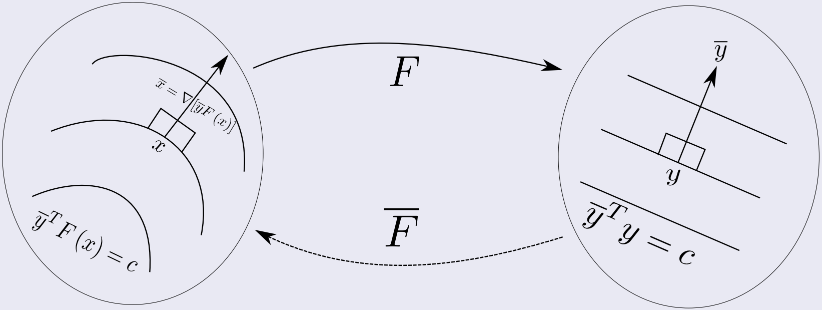
- Thus, we have a dual process

Outline

Cinvestav

Dual Process

Here, we have that the hyperplane $\overline{y}^T\overline{y} = c$ in the range of F has inverse image $\left\{x|\overline{y}^T F(x) = c\right\}$



The implicit function theorem

Theorem

- Let $F : \mathbb{R}^{n+m} \to \mathbb{R}^m$ be a continuously differentiable function, and a point $\left(x_1^0, x_2^0, ..., x_{m+n}^0\right)$ so $F\left(x_1^0, x_2^0, ..., x_{m+n}^0\right) = c$. If $\frac{\partial F\left(x_1^0, x_2^0, ..., x_{m+n}^0\right)}{\partial x_{m+n}} \neq 0$, then there exist a neighborhood of $\left(x_1^0, x_2^0, ..., x_{m+n}^0\right)$ so whatever $\left(x_1, ..., x_{n+m-1}\right)$ is close enough to $\left(x_1^0, ..., x_{m+n-1}^0\right)$, there is a unique z so that $F\left(x_1, ..., x_{n+m-1}, z\right) = c$. Furthermore, $z = g\left(x_1, ..., x_{n+m-1}\right)$ a continuous function of $\left(x_1, ..., x_{n+m-1}\right)$.

Cinvestav

Therefore

The set $\left\{x|\overline{y}^T F\left(x\right)=c\right\}$

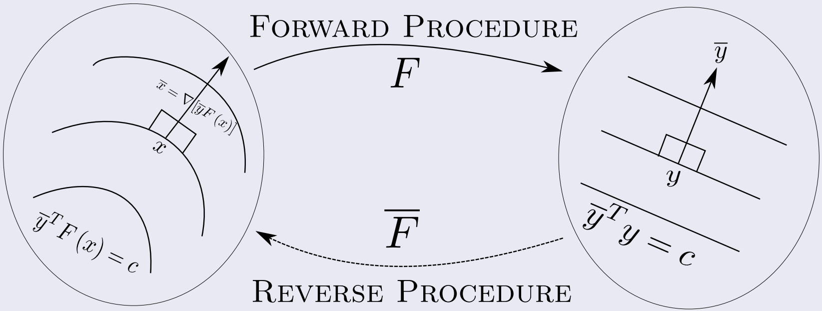
- It is a smooth hyper-surface with the normal

$$\overline{x}^T=\overline{y}^T F'\left(x\right)$$

at x provided that $\overline{x}$ does not vanishes.

The Process

Here, we have that the hyperplane $\overline{y}^T\overline{y} = c$ in the range of F has inverse image $\left\{x|\overline{y}^T F(x) = c\right\}$



Therefore

When $m = 1$, then $F = f$ is scaler-valued

- We obtain $\overline{y} = 1 \in \mathbb{R}$ the familiar gradient $\nabla f\left(x\right) = \overline{y}^T F'\left(x\right)$.

Therefore

When $m = 1$, then $F = f$ is scaler-valued

- We obtain $\overline{y} = 1 \in \mathbb{R}$ the familiar gradient $\nabla f(x) = \overline{y}^T F'(x)$.

Something Notable

- We will look only at the main procedure of Incremental Adjoint Recursion

Therefore

When $m = 1$, then $F = f$ is scaler-valued

- We obtain $\overline{y} = 1 \in \mathbb{R}$ the familiar gradient $\nabla f(x) = \overline{y}^T F'(x)$.

Something Notable

- We will look only at the main procedure of Incremental Adjoint Recursion

Please take a look at section in **Derivation by Matrix-Product Reversal**

- At the book [5]
  - Andreas Griewank and Andrea Walther, **Evaluating derivatives: principles and techniques of algorithmic differentiation** vol. 105, (Siam, 2008).

Cinvestav

The derivation of the reversal mode

For this, we will use

$v_{i-n} \equiv x_i$ $\dot{v}_{i-n} \equiv \dot{x}_i$	$i = 1 \ldots n$
$v_i \equiv \phi_i \left(v_j\right)_{j \prec i} \; i = 1 \ldots l$ $\dot{v}_i \equiv \sum_{j \prec i} \frac{\partial \phi_i (u_j)}{\partial v_j} \dot{v}_j$	$i = 1 \ldots l$
$y_{m-i} \equiv v_{l-i}$ $\dot{y}_{m-i} \equiv \dot{v}_{l-i}$	$i = m - 1 \ldots 0$

The derivation of the reversal mode

For this, we will use

$v_{i-n} \equiv x_i$ $\dot{v}_{i-n} \equiv \dot{x}_i$	$i = 1 \ldots n$
$v_i \equiv \phi_i \left(v_j\right)_{j \prec i} \; i = 1 \ldots l$ $\dot{v}_i \equiv \sum_{j \prec i} \frac{\partial \phi_i(u_j)}{\partial v_j} \dot{v}_j$	$i = 1 \ldots l$
$y_{m-i} \equiv v_{l-i}$ $\dot{y}_{m-i} \equiv \dot{v}_{l-i}$	$i = m-1 \ldots 0$

And the identity to find $\overline{x}$

$$\overline{y}^T \dot{y} = \overline{x}^T \dot{x}$$

Now, using the state transformation Φ

We map from x to $y = F(x)$ as the composition

$$y = Q_m \Phi_l \circ \Phi_{l-1} \circ \cdots \circ \Phi_2 \circ \Phi_1 \left(P_n^T x \right)$$

- Where $P_n \equiv [I, 0, ..., 0] \in \mathbb{R}^{n \times (n+l)}$ and $Q_m \equiv [0, 0, ..., I] \in \mathbb{R}^{m \times (n+l)}$

Cinvestav

Now, using the state transformation Φ

We map from x to $y = F(x)$ as the composition

$$y = Q_m \Phi_l \circ \Phi_{l-1} \circ \cdots \circ \Phi_2 \circ \Phi_1 \left(P_n^T x\right)$$

- Where $P_n \equiv [I, 0, ..., 0] \in \mathbb{R}^{n \times (n+l)}$ and $Q_m \equiv [0, 0, ..., I] \in \mathbb{R}^{m \times (n+l)}$

They are matrices that project an arbitrary $(n+l)$-vector

- Onto its first n and last m components (Or input to output if you please)

Where

The c_{ij}'s represent partial differential

$$c_{ij} \equiv c_{ij}\left(u_i\right) \equiv \frac{\partial \phi_i}{\partial v_j} \text{ for } 1-n \leq i,j \leq l$$

Labeling the elemental partials as c_{ij}

We get the state Jacobian

$$A_i \equiv \Phi_i' \equiv \begin{bmatrix} 1 & 0 & \dots & 0 & \dots & \dots & 0 \\ 0 & 1 & \dots & 0 & \dots & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \dots & \dots & \\ 0 & 0 & \dots & 1 & \dots & \dots & 0 \\ c_{i1-n} & c_{i2-n} & \dots & c_{ii-n} & \dots & \dots & 0 \\ 0 & 0 & \dots & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & \dots & \dots & \dots & 1 \end{bmatrix} \in \mathbb{R}^{(n+l)\times(n+l)}$$

- where the c_{ij} occur in the $(n+i)$th row of A_i.

Cinvestav

Remarks

The square matrices A_i are lower triangular

- It may also be written as rank-one perturbations of the identity,

$$A_i = I + e_{n+i} \left[\nabla \phi_i \left(u_i \right) - e_{n+i} \right]^T$$

  - Where e_j denotes the jth Cartesian basis vector in $\mathbb{R}^{n+l}$

Cinvestav

Remarks

The square matrices A_i are lower triangular

- It may also be written as rank-one perturbations of the identity,

$$A_i = I + e_{n+i}\left[\nabla\phi_i\left(u_i\right) - e_{n+i}\right]^T$$

  - Where e_j denotes the jth Cartesian basis vector in $\mathbb{R}^{n+l}$

The differentiating the composition of functions, we get

$$\dot{y} = Q_m A_l A_{l-1} \cdots A_2 A_1 P_n^T \dot{x}$$

Embeddings

The multiplication by $P_n^T \in \mathbb{R}^{(n+l)\times n}$

- It embeds $\dot{x}$ into $\mathbb{R}^{n+l}$, a Projection

Embeddings

The multiplication by $P_n^T \in \mathbb{R}^{(n+l)\times n}$

- It embeds $\dot{x}$ into $\mathbb{R}^{n+l}$, a Projection

Meaning

- orresponding to the first part of the tangent recursion

Embeddings

The multiplication by $P_n^T \in \mathbb{R}^{(n+l)\times n}$

- It embeds $\dot{x}$ into $\mathbb{R}^{n+l}$, a Projection

Meaning

- orresponding to the first part of the tangent recursion

The subsequent multiplications by the A_i

- It generates ine component $\dot{v}_i$ at a time, according to the middle part

Finally

Q_m extracts the last m components as $\dot{y}$ corresponding to the third part of the table

$v_{i-n} \equiv x_i$ $\dot{v}_{i-n} \equiv \dot{x}_i$	$i = 1...n$
$v_i \equiv \phi_i \left(v_j\right)_{j \prec i}$ $i = 1...l$ $\dot{v}_i \equiv \sum_{j \prec i} \frac{\partial \phi_i(u_j)}{\partial v_j} \dot{v}_j$	$i = 1...l$
$y_{m-i} \equiv v_{l-i}$ $\dot{y}_{m-i} \equiv \dot{v}_{l-i}$	$i = m-1...0$

Cinvestav

Now

By comparison with

$$\dot{y}(t) = \frac{\partial F(x(t))}{\partial t} = F'(x(t))\,\dot{x}(t)$$

Now

By comparison with

$$\dot{y}(t) = \frac{\partial F(x(t))}{\partial t} = F'(x(t))\,\dot{x}(t)$$

We have in fact a product representation of the full Jacobian

$$F'(x) = Q_m A_l A_{l-1} \cdots A_2 A_1 P_n^T \in \mathbb{R}^{m \times n}$$

Then

By transposing the product we obtain the adjoint relation

$$\overline{x} = P_n A_1^T A_2^T \cdots A_{l-1}^T A_l^T \overline{y}$$

Then

By transposing the product we obtain the adjoint relation

$$\overline{x} = P_n A_1^T A_2^T \cdots A_{l-1}^T A_l^T \overline{y}$$

Given that

$$A_i^T = I + \left[\nabla \phi_i \left(u_i\right) - e_{n+i}\right] e_{n+i}^T$$

Therefore

The transformation of any vector $(\overline{v}_j)_{1-n \leq j \leq l}$

- By multiplication with A_i^T representing an incremental operation.

In detail, one obtains for $i = l, ..., 1$ the operations

For all j with $i \neq j \not< i$

- $\overline{v}_j$ is left unchanged

In detail, one obtains for $i = l, \ldots, 1$ the operations

For all j with $i \neq j \not\prec i$

- $\overline{v}_j$ is left unchanged

For all j with $i \neq j \prec i$

- $\overline{v}_i$ is augmented by $\overline{v}_i c_{ij}$

$$c_{ij} \equiv c_{ij}\left(u_i\right) \equiv \frac{\partial \phi_i}{\partial v_j} \text{ for } 1 - n \leq i, j \leq l$$

In detail, one obtains for $i = l, ..., 1$ the operations

For all j with $i \neq j \nprec i$

- $\overline{v}_j$ is left unchanged

For all j with $i \neq j \prec i$

- $\overline{v}_i$ is augmented by $\overline{v}_i c_{ij}$

$$c_{ij} \equiv c_{ij}\left(u_i\right) \equiv \frac{\partial \phi_i}{\partial v_j} \text{ for } 1 - n \leq i, j \leq l$$

Subsequently

- $\overline{v}_i$ is set to zero.

Outline

Cinvestav

Some Remarks

Using the C-style abbreviation

- $a+ \equiv b$ for $a \equiv a + b$
  - We may rewrite the matrix- vector product as the adjoint evaluation procedure in the following table

Incremental Adjoint Recursion

We have the following procedure $\left(u_i = (v_j)_{j \prec i} \in \mathbb{R}^{n_i}\right)$

$\overline{v}_i \equiv 0$	$i = 1 - n \ldots l$
$\overline{v}_{i-n} \equiv x_i$	$i = 1 \ldots n$
$v_i \equiv \phi_i \left(v_j\right)_{j \prec i}$	$i = m - 1 \ldots l$
$y_{m-i} \equiv v_{l-i}$	$i = 0 \ldots m - 1$

$\overline{v}_{l-i} \equiv \overline{y}_{m-i}$	$i = 0 \ldots m - 1$
$\overline{v}_j + \equiv \overline{v}_i \frac{\partial \phi_i(u_i)}{\partial v_j}$ for $j \prec i$	$i = l \ldots 1$
$\overline{x}_i \equiv \overline{v}_{i-n}$	$i = n \ldots 1$

Cinvestav

Explanation

It is assumed as a precondition that the adjoint quantities

- $\overline{v}_i$ for $1 \leq i \leq l$ have been initialized to zero

Explanation

It is assumed as a precondition that the adjoint quantities

- $\overline{v}_i$ for $1 \leq i \leq l$ have been initialized to zero

As indicated by the range specification $i = l, ..., 1$

- we think of the incremental assignments as being executed in reverse order, i.e., for $i = l, l-1, l-2, ..., 1$.

Explanation

It is assumed as a precondition that the adjoint quantities

- $\overline{v}_i$ for $1 \leq i \leq l$ have been initialized to zero

As indicated by the range specification $i = l, ..., 1$

- we think of the incremental assignments as being executed in reverse order, i.e., for $i = l, l-1, l-2, ..., 1$.

Only then is it guaranteed

- Each $\overline{v}_i$ will reach its full value before it occurs on the right-hand side.

Cinvestav

Furthermore

We can combine the incremental operations

- Affected by the adjoint of ϕ_i to

$$\overline{u}_i {+}= \overline{v}_i \cdot \nabla \phi_i \left(u_i \right) \text{ where } \overline{u}_i \equiv \left(\overline{u}_j \right)_{j \prec i} \in \mathbb{R}^{n_i}$$

Cinvestav

Furthermore

We can combine the incremental operations

- Affected by the adjoint of ϕ_i to

$$\overline{u}_i + = \overline{v}_i \cdot \nabla \phi_i \left(u_i \right) \text{ where } \overline{u}_i \equiv \left(\overline{u}_j \right)_{j \prec i} \in \mathbb{R}^{n_i}$$

Something Remarkable

- We can do something different
  - one can directly compute the value of the adjoint quantity $\overline{v}_j$ by collecting all contributions to it as a sum ranging over all successors $i \succ j$.

Cinvestav

Furthermore

We can combine the incremental operations

- Affected by the adjoint of ϕ_i to

$$\overline{u}_i + = \overline{v}_i \cdot \nabla\phi_i\left(u_i\right) \text{ where } \overline{u}_i \equiv \left(\overline{u}_j\right)_{j \prec i} \in \mathbb{R}^{n_i}$$

Something Remarkable

- We can do something different
  - one can directly compute the value of the adjoint quantity $\overline{v}_j$ by collecting all contributions to it as a sum ranging over all successors $i \succ j$.

This no-incremental

- Requires global information that is not easy to come by.

Complexity

Something Notable

$$TIME\left\{F\left(x\right),\overline{y}^{T}F'\left(x\right)\right\}\leq w_{grad}TIME\left\{F\left(x\right)\right\}$$

- Where $w_{grad}\in[3,4]$ (The cheap gradient principle)

Remember

Time Complexity

$$TIME\left\{F\left(x\right), F'\left(x\right)\dot{x}\right\} \leq w_{tan} TIME\left\{F\left(x\right)\right\}$$

- Where $w_{tan} \in \left[2, \frac{5}{2}\right]$

Cinvestav

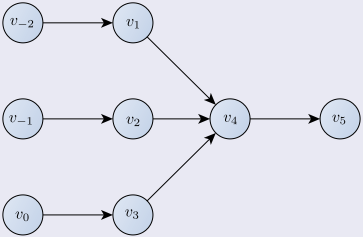
Outline

Cinvestav

Example a single layer perceptron

We have

$$y = \sigma\left(\sum_{i=1}^{3} w_i x_i\right)$$



First Phase

Forward Step

Forward Step
$v_{-2} = w_1$
$v_{-1} = w_2$
$v_0 = w_3$
$v_1 = x_1 v_{-2}$
$v_2 = x_2 v_{-1}$
$v_3 = x_3 v_0$
$v_4 = v_1 + v_2 + v_3$
$v_5 = \sigma\left(v_4\right)$
$y_1 = v_5$

Cinvestav

Second Phase

Incremental Return

Forward Step
$v_{-2} = w_1$
$v_{-1} = w_2$
$v_0 = w_3$
$v_1 = x_1 v_{-2}$
$v_2 = x_2 v_{-1}$
$v_3 = x_3 v_0$
$v_4 = v_1 + v_2 + v_3$
$v_5 = \sigma\left(v_4\right)$
$y_1 = v_5$

Incremental Return
$\overline{v}_5 = \overline{y}_1 = 1$
$\overline{v}_4 = \frac{\partial v_5}{\partial v_4}\overline{y}_1 = \sigma'\left(v_4\right)$
$\overline{v}_3 + = \frac{\partial v_4}{\partial v_3}\overline{v}_4 = 1 \times \sigma'\left(v_4\right)$
$\overline{v}_0 = \frac{\partial v_3}{\partial v_0}\overline{v}_3 = x_3 \times \sigma'\left(v_4\right)$
$\overline{v}_2 + = \frac{\partial v_4}{\partial v_2}\overline{v}_4 = 1 \times \sigma'\left(v_4\right)$
$\overline{v}_{-1} = \frac{\partial v_2}{\partial v_{-1}}\overline{v}_2 = x_2 \times \sigma'\left(v_4\right)$
$\overline{v}_1 + = \frac{\partial v_4}{\partial v_1}\overline{v}_4 = 1 \times \sigma'\left(v_4\right)$
$\overline{v}_{-2} = \frac{\partial v_1}{\partial v_{-2}}\overline{v}_1 = x_1 \times \sigma'\left(v_4\right)$
$\overline{w}_3 = x_3 \times \sigma'\left(v_4\right)$
$\overline{w}_2 = x_2 \times \sigma'\left(v_4\right)$
$\overline{w}_1 = x_1 \times \sigma'\left(v_4\right)$

How does it compares with the Forward Mode?

We noticed that you do the following for each gradient variable

Forward Step; Gradient of Forward Step
$v_{-2} = w_1; \dot{v}_{-2} = \dot{w}_1 = 0$
$v_{-1} = w_2; \dot{v}_{-1} = \dot{w}_2 = 0$
$v_0 = w_3; \dot{v}_0 = \dot{w}_2 = 1$
$v_1 = x_1 v_{-2}$
$\dot{v}_1 = x_1 \dot{v}_{-2} = 0$
$v_2 = x_2 v_{-1}$
$\dot{v}_2 = x_2 \dot{v}_{-1} = 0$
$v_3 = w_3 v_0$
$\dot{v}_3 = x_3 \dot{v}_0 = x_3$
$v_4 = v_1 + v_2 + v_3$
$\dot{v}_4 = \dot{v}_1 + \dot{v}_2 + \dot{v}_3 = x_3$
$v_5 = \sigma\left(v_4\right)$
$\dot{v}_5 = \dot{v}_4 = x_3 \times \sigma'\left(v_4\right)$
$y_1 = v_5; \dot{y}_1 = \dot{v}_5$

Outline

Cinvestav

Let us to look at the following example

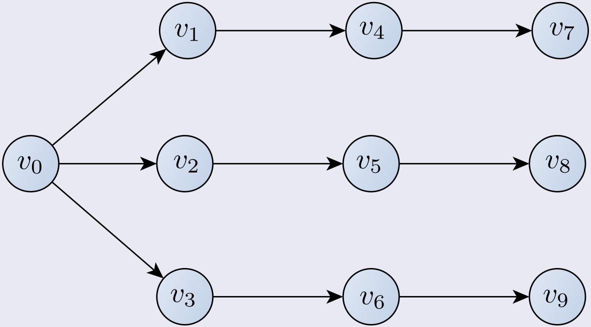
We have the following system of equations

$$
\begin{aligned}
y_1 &= \sigma\left(w_1 x\right)\\
y_2 &= \sigma\left(w_2 x\right)\\
y_3 &= \sigma\left(w_2 x\right)
\end{aligned}
$$

Cinvestav

With the following graph

Notice the difference with a neural network



The Forward mode looks like

We have that

$$
\begin{array}{|c|}
\hline
v_0 = x; \dot{v}_0 = \dot{x} = 1 \\
\hline
\end{array}
$$

$$
\begin{array}{|c|}
\hline
v_1 = w_1 v_0 \\
\hline
\dot{v}_1 = w_1 \dot{v}_{-2} = w_1 \\
\hline
v_2 = w_2 v_0 \\
\hline
\dot{v}_2 = w_1 \dot{v}_0 = w_2 \\
\hline
v_3 = w_3 v_0 \\
\hline
\dot{v}_3 = w_1 \dot{v}_0 = w_3 \\
\hline
v_4 = \sigma\left(v_1\right) \\
\hline
\dot{v}_4 = \sigma'\left(v_1\right) \times \dot{v}_1 = w_1 \times \sigma'\left(v_1\right) \\
\hline
v_5 = \sigma\left(v_2\right) \\
\hline
\dot{v}_5 = \sigma'\left(v_2\right) \times \dot{v}_2 = w_2 \times \sigma'\left(v_2\right) \\
\hline
v_6 = \sigma\left(v_3\right) \\
\hline
\dot{v}_6 = \sigma'\left(v_3\right) \times \dot{v}_3 = w_3 \times \sigma'\left(v_3\right) \\
\hline
\end{array}
\Longrightarrow
\begin{array}{|c|}
\hline
y_1 = v_4; \dot{y}_1 = \dot{v}_4 \\
\hline
y_2 = v_5; \dot{y}_2 = \dot{v}_5 \\
\hline
y_3 = v_6; \dot{y}_3 = \dot{v}_6 \\
\hline
\end{array}
$$

Cinvestav

Now you can see it

Forward and Reverse Mode

- They depend on the input and output size!!!

Now you can see it

Forward and Reverse Mode

- They depend on the input and output size!!!

A More Formal Definition

- For a function $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$, suppose we wish to compute all the elements of the $m \times n$ Jacobian matrix

Cinvestav

Now you can see it

Forward and Reverse Mode

- They depend on the input and output size!!!

A More Formal Definition

- For a function $f : \mathbb{R}^n \to \mathbb{R}^m$, suppose we wish to compute all the elements of the $m \times n$ Jacobian matrix

Ignoring the overhead of building the expression graph

- Under this situation Reverse Mode requires m sweeps performs better when $n > m$.

Consequences for Deep Learning

With a relatively small overhead

- The performance of reverse-mode AD is superior when $n \gg m$, that is when we have many inputs and few outputs.

Consequences for Deep Learning

With a relatively small overhead

- The performance of reverse-mode AD is superior when $n \gg m$, that is when we have many inputs and few outputs.

As we saw it in the previous examples

- If $n \ll m$ forward mode performs better

Special Cases

Nevertheless when we have a comparable number of outputs and inputs

- Forward mode can be more efficient,
  - less overhead associated with storing the expression graph in memory in forward mode.

Special Cases

Nevertheless when we have a comparable number of outputs and inputs

- Forward mode can be more efficient,
  - less overhead associated with storing the expression graph in memory in forward mode.

For Example

- If you have $f:\mathbb{R}^n \to \mathbb{R}$, when $n=1$ forward mode is more efficient, but the result flips as n increases.

Outline

Cinvestav

We have the followowing

Something Notable

Forward Mode Automatic Differentiation	$\Leftrightarrow$	**Dual Number Function Evaluation**

Dual Numbers

In algebra, the dual numbers are a hypercomplex number system

- They are expressions of the form $a + b\epsilon$ where $\epsilon > 0$ and $\epsilon^2 = 0$

Dual Numbers

In algebra, the dual numbers are a hypercomplex number system

- They are expressions of the form $a + b\epsilon$ where $\epsilon > 0$ and $\epsilon^2 = 0$

Dual numbers can be added component-wise

- $(a + b\epsilon) + (c + d\epsilon) = a + c + (b + d)\,\epsilon$
- In addition, $(a + b\epsilon)(c + d\epsilon) = ac + (ad + bc)\,\epsilon$

Cinvestav

Dual Numbers

In algebra, the dual numbers are a hypercomplex number system

- They are expressions of the form $a + b\epsilon$ where $\epsilon > 0$ and $\epsilon^2 = 0$

Dual numbers can be added component-wise

- $(a + b\epsilon) + (c + d\epsilon) = a + c + (b + d)\,\epsilon$
- In addition, $(a + b\epsilon)(c + d\epsilon) = ac + (ad + bc)\,\epsilon$

Actually

- This is actually very similar to the idea of a complex number

Cinvestav

We also have the division of dual numbers

For example, when $c \neq 0$

$$\begin{aligned}
\frac{a+b\epsilon}{c+d\epsilon} &= \frac{(a+b\epsilon)(c-d\epsilon)}{(c+d\epsilon)(c-d\epsilon)} \\
&= \frac{ac-ad\epsilon+bc\epsilon-bd\epsilon^2}{c^2+cd\epsilon-cd\epsilon-d^2\epsilon^2} \\
&= \frac{ac-ad\epsilon+bc\epsilon}{c^2} \\
&= \frac{a}{c}+\frac{bc-ad}{c^2}\epsilon
\end{aligned}$$

Dual numbers to the problem of calculating the derivative of a function

We can add an infinitesimal quantity to each side of the equation

$$
\begin{aligned}
y =& f\left(x\right) \\
y + \frac{\partial y}{\partial x} dx =& f\left(x\right) + f'\left(x\right) dx
\end{aligned}
$$

Dual numbers to the problem of calculating the derivative of a function

We can add an infinitesimal quantity to each side of the equation

$$
\begin{aligned}
y =& f(x) \\
y + \frac{\partial y}{\partial x} dx =& f(x) + f'(x) dx
\end{aligned}
$$

Such that the derivative $f(x)' = \frac{\partial y}{\partial x}$

- It is the one we want.

Cinvestav

Given that for infinitesimal numbers dx

The function is linear in a small area

$$f\left(x+dx\right)=f\left(x\right)+f'\left(x\right)dx$$

Given that for infinitesimal numbers dx

The function is linear in a small area

$$f\left(x+dx\right)=f\left(x\right)+f'\left(x\right)dx$$

Now, the Chain Rule - Backpropagation

$$\begin{aligned} f\left(g\left(x+dx\right)\right) =& f\left(g\left(x\right)+g'\left(x\right)dx\right) \\ =& f\left(g\left(x\right)\right)+f'\left(g\left(x\right)\right)g'\left(x\right)dx \end{aligned}$$

Meaning

Something Notable

- This means that we can easily propagate gradients across the layers of computation simply be multiplying derivatives with each other.

Meaning

Something Notable

- This means that we can easily propagate gradients across the layers of computation simply be multiplying derivatives with each other.

Therefore if we assume an input is $x = v + \dot{v}dx$

- To implement the dual numbers we simply require a separate storage systems that keeps track of $x = v$ coefficient in front of $\dot{v}$
- And apply the respective derivative computations to the infinitesimal part of x

We can then use the dual's

Instead of using dx, we can use ϵ for our i variables and $\dot{v}$ the derivative

$$x = v + \dot{v}\epsilon$$

We can then use the dual's

Instead of using dx, we can use ϵ for our i variables and $\dot{v}$ the derivative

$$x = v + \dot{v}\epsilon$$

Example on the the function $f(x) = 3x + 2$

- We want to calculate $f(4)$ and $f'(4)$

Thus, we can do the following

We convert the 4 into a dual form $4 + 1\epsilon$

1. $(4 + 1\epsilon)(3 + 0\epsilon) = 12 + 0\epsilon + 3\epsilon + 0\epsilon^2 = 12 + 3\epsilon$
2. $(12 + 3\epsilon) + (2 + 0\epsilon) = 14 + 3\epsilon$

Cinvestav

Thus, we can do the following

We convert the 4 into a dual form $4 + 1\epsilon$

1. $(4 + 1\epsilon)(3 + 0\epsilon) = 12 + 0\epsilon + 3\epsilon + 0\epsilon^2 = 12 + 3\epsilon$
2. $(12 + 3\epsilon) + (2 + 0\epsilon) = 14 + 3\epsilon$

Something Notable

1. $f(4) = 14$
2. $f'(4) = 3$

Outline

Cinvestav

There is an isomorphism into the 2×2 matrices

Basically

$$a + b\epsilon \leftrightarrow \begin{pmatrix} a & b \\ 0 & a \end{pmatrix}$$

There is an isomorphism into the 2×2 matrices

Basically

$$a + b\epsilon \leftrightarrow \begin{pmatrix} a & b \\ 0 & a \end{pmatrix}$$

Therefore, we have for example

$$\begin{pmatrix} a & b \\ 0 & a \end{pmatrix} \begin{pmatrix} c & d \\ 0 & c \end{pmatrix} = \begin{pmatrix} ac & ad + bc \\ 0 & ac \end{pmatrix} \leftrightarrow ac + (ad + bc)\,\epsilon$$

Cinvestav

There is an isomorphism into the 2×2 matrices

Basically

$$a + b\epsilon \leftrightarrow \begin{pmatrix} a & b \\ 0 & a \end{pmatrix}$$

Therefore, we have for example

$$\begin{pmatrix} a & b \\ 0 & a \end{pmatrix} \begin{pmatrix} c & d \\ 0 & c \end{pmatrix} = \begin{pmatrix} ac & ad + bc \\ 0 & ac \end{pmatrix} \leftrightarrow ac + (ad + bc)\,\epsilon$$

Finally

$$\epsilon \leftrightarrow \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$$

Then, we can the Matrix definition for representation

In the multivariate case

$$x = v + \dot{v}\epsilon$$

$$y = u + \dot{u}\epsilon$$

Then, we can the Matrix definition for representation

In the multivariate case

$$x = v + \dot{v}\epsilon$$

$$y = u + \dot{u}\epsilon$$

Thus, the partial derivative $\frac{\partial x}{\partial x}$

- First we have the matrix representation

$$M_x = \begin{pmatrix} v & \dot{v} \\ 0 & v \end{pmatrix}$$

Therefore

We have that

$$\frac{\partial M_x}{\partial v} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Therefore

We have that

$$\frac{\partial M_x}{\partial v} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Furthermore, we have that

$$\frac{\partial M_x}{\partial \dot{v}} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$$

Outline

We can try to simply implement a Regression

Something as using the Cross Entropy over Logistic

$$L \circ \sigma (X, y, \beta) = y \log \sigma (X\beta) + (1 - y) \log (1 - \sigma (X\beta))$$

Then, How do we implement this?

First, the Dual Tensor

- class DualTensor(object):
- # Class object for dual representation of a tensor/matrix/vector
- def __init__(self, real, dual):
- self.real = real
- self.dual = dual # The infinitesimal part
- def zero_grad(self):
- # Reset the gradient for the next batch evaluation
- dual_part = np.zeros((len(self.real), len(self.real)))
- np.fill_diagonal(dual_part, 1)
- self.dual = dual_part
- return

Addition

Adding the dual numbers

```
def add_duals(dual_a, dual_b):
    # Operator non-"overload": Add a two dual numbers
    real_part = dual_a.real + dual_b.real
    dual_part = dual_a.dual + dual_b.dual
    return DualTensor(real_part, dual_part)
```

Cinvestav

Now, the Dot Product

We have

$$x = a + b\epsilon$$
$$y = c + d\epsilon$$

Now, the Dot Product

We have

$$x = a + b\epsilon$$
$$y = c + d\epsilon$$

We have for the dot product $x \cdot y$ of two vectors

$$\begin{aligned} x \cdot y =& (a + b\epsilon) \cdot (c + d\epsilon) \\ =& a \cdot c + b \cdot c\epsilon + a \cdot d\epsilon + b \cdot d\epsilon^2 \\ =& a \cdot c + b \cdot c\epsilon + a \cdot d\epsilon \end{aligned}$$

Now, the Dot Product

We have

$$x = a + b\epsilon$$
$$y = c + d\epsilon$$

We have for the dot product $x \cdot y$ of two vectors

$$\begin{aligned} x \cdot y =& (a + b\epsilon) \cdot (c + d\epsilon) \\ =& a \cdot c + b \cdot c\epsilon + a \cdot d\epsilon + b \cdot d\epsilon^2 \\ =& a \cdot c + b \cdot c\epsilon + a \cdot d\epsilon \end{aligned}$$

Therefore, if we multiply against a vector with no gradient as

$$x \cdot y = a \cdot c + a \cdot d\epsilon$$

Now

Dot Product

- def dot_product(b_dual, x, both_require_grad=False):
- # Function to perform dot product between a dual and a no grad_req vector
- real_part = np.dot(x.real, b_dual.real) $\#a \cdot c$
- dual_part = np.dot(x.real, b_dual.dual) $\#a \cdot d\epsilon$
- if both_require_grad:
- dual_part += np.dot(b_dual.real, x.dual) $\#\ b \cdot c\epsilon$
- return DualTensor(real_part, dual_part)

What about the Log?

We have that the log of a dual number z composed by a real part and the dual part

$$\log z = \log x + \frac{y}{x}\epsilon$$

Cinvestav

What about the Log?

We have that the log of a dual number z composed by a real part and the dual part

$$\log z = \log x + \frac{y}{x}\epsilon$$

This is because a dual number is written as $z = x + y\epsilon$

- Then, we have
  $\log(x + y\epsilon) = \log\left(x\left[1 + \frac{y}{x}\epsilon\right]\right) = \log(x) + \log\left(1 + \frac{y}{x}\epsilon\right)$

Cinvestav

For this, we can use the Taylor expansion

The Taylor series for $\log(1+x)$ around

- Then, we have
  $\log(x+y\epsilon) = \log\left(x\left[1+\frac{y}{x}\epsilon\right]\right) = \log(x) + \log\left(1+\frac{y}{x}\epsilon\right)$

For this, we can use the Taylor expansion

The Taylor series for $\log(1+x)$ around

- Then, we have
  $\log(x+y\epsilon) = \log\left(x\left[1+\frac{y}{x}\epsilon\right]\right) = \log(x) + \log\left(1+\frac{y}{x}\epsilon\right)$

We know that

- Then, we know that $\log(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \cdots$

Cinvestav

For this, we can use the Taylor expansion

The Taylor series for $\log(1+x)$ around

- Then, we have
  $\log(x + y\epsilon) = \log\left(x\left[1 + \frac{y}{x}\epsilon\right]\right) = \log(x) + \log\left(1 + \frac{y}{x}\epsilon\right)$

We know that

- Then, we know that $\log(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \cdots$

Then, we have that

- $\log\left(1 + \frac{y}{x}\epsilon\right) = \frac{y}{x}\epsilon$

Finally, we have

We have that

- $\log(x + y\epsilon) = \log(x) + \log\left(1 + \frac{y}{x}\epsilon\right) = \log(x) + \frac{y}{x}\epsilon$

Log on Dual Tensor

We have

- def log(dual_tensor):
-         # Operator non-"overload": Log (real) & its derivative (dual)
-         real_part = np.log(dual_tensor.real)
-         temp_1 = 1/dual_tensor.real
-         # Fill matrix with diagonal entries of log derivative
-         temp_2 = np.zeros((temp_1.shape[0], temp_1.shape[0]))
-         np.fill_diagonal(temp_2, temp_1)
-         dual_part = np.dot(temp_2, dual_tensor.dual)
-         return DualTensor(real_part, dual_part)

Now the sigmoid

First remember how to derive the sigmoid function

$$f\left(g\left(x\right)\right)=\frac{1}{1+\exp\left\{-g\left(x\right)\right\}}$$

Now the sigmoid

First remember how to derive the sigmoid function

$$f\left(g\left(x\right)\right)=\frac{1}{1+\exp\left\{-g\left(x\right)\right\}}$$

We have the following

$$\nabla f\left(g\left(x\right)\right)=\left(\frac{1}{1+\exp\left\{-g\left(x\right)\right\}}\right)\left(1-\frac{1}{1+\exp\left\{-g\left(x\right)\right\}}\right)\nabla g\left(x\right)$$

Thus, we have that

Something Notable

- def sigmoid(dual_tensor):
- # Operator non-"overload": Sigmoid (real) & its derivative (dual)
- real_part = 1/(1+np.exp(-dual_tensor.real))
- temp_1 = np.multiply(real_part, 1-real_part)
- # Fill matrix with diagonal entries of sigmoid derivative
- temp_2 = np.zeros((temp_1.shape[0], temp_1.shape[0]))
- np.fill_diagonal(temp_2, temp_1)
- dual_part = np.dot(temp_2, dual_tensor.dual)
- return DualTensor(real_part, dual_part)

Cost function

the Cross Entropy over Logistic

$$L \circ \sigma(X, y, \beta) = y \log \sigma(X\beta) + (1 - y) \log(1 - \sigma(X\beta))$$

How the Forward Looks

Forward

- def forward(X, b_dual):
- \# Apply element-wise sigmoid activation
- y_pred_1 = sigmoid(dot_product(b_dual, X))
- y_pred_2 = DualTensor(1-y_pred_1.real, -y_pred_1.dual)
- \# Make numerically stable!
- y_pred_1.real = np.clip(y_pred_1.real, 1e-15, 1-1e-15)
- y_pred_2.real = np.clip(y_pred_2.real, 1e-15, 1-1e-15)
- return y_pred_1, y_pred_2

Cinvestav

Now, binary cross entropy dual

We have

- def binary_cross_entropy_dual(y_true, y_pred_1, y_pred_2):
-     # Compute actual binary cross-entropy term
-     log_y_pred_1, log_y_pred_2 = log(y_pred_1), log(y_pred_2)
-     bce_l1, bce_l2 = dot_product(log_y_pred_1, -y_true), dot_product(log_y_pred_2, -(1 -y_true)
-     bce = add_duals(bce_l1, bce_l2)
-     # Calculate the batch classification accuracy
-     acc = (y_true == (y_pred_1.real > 0.5)).sum()/y_true.shape[0]
-     return bce, acc

Cinvestav

In pytorch

We have a extra step in the batch training

- We have the following line
  - optimizer.zero_grad()

In pytorch

We have a extra step in the batch training

- We have the following line
  - optimizer.zero_grad()

Yes, it is the preparation for the use of dual numbers or something fancier

- As they say... WOW

Thus, we have that

Something Notable

- def zero_grad(self):
- # Reset the gradient for the next batch evaluation
- dual_part = np.zeros((len(self.real), len(self.real)))
- np.fill_diagonal(dual_part, 1)
- return dual_part return

Train the stuff

We have

- def train_logistic_regression(n, d, n_epoch, batch_size, b_init, l_rate):
- # Generate the data for a coefficient vector & init progress tracker!
- data_loader = DataLoader(n, d, batch_size, binary=True)
- b_dual = DualTensor(b_init, None)
- # Start running the training loop
- for epoch in range(n_epoch):
- data_loader.shuffle_arrays()
- for batch_id in range(data_loader.num_batches):
- # Clear the gradient
- b_dual.zero_grad()
- # Select the current batch & perform "mini-forward" pass
- X, y = data_loader.get_batch_idx(batch_id)
- y_pred_1, y_pred_2 = forward(X, b_dual)
- # Calculate the forward AD - real = func, dual = deriv
- current_dual, acc = binary_cross_entropy_dual(y, y_pred_1, y_pred_2)
- # Perform grad step & append results to the placeholder list
- b_dual.real -= l_rate*np.array(current_dual.dual).flatten()

Outline

Cinvestav

Between Two Extremes

Something Notable

- Forward and reverse accumulation are just two (extreme) ways of traversing the chain rule.

Between Two Extremes

Something Notable

- Forward and reverse accumulation are just two (extreme) ways of traversing the chain rule.

The problem of computing a full Jacobian of $f : \mathbb{R}^n \to \mathbb{R}^m$ with a minimum number of arithmetic operations

- It is known as the Optimal Jacobian Accumulation (OJA) problem, which is NP-complete [10].

Finally

Using all the previous ideas

- The Graph Structure Proposed in [11]
- The Computational Graph of AD
- The Forward and Reversal Methods

Finally

Using all the previous ideas

- The Graph Structure Proposed in [11]
- The Computational Graph of AD
- The Forward and Reversal Methods

It has been possible to develop the Deep Learning Frameworks

- TensorFlow
- Torch
- Pytorch
- Keras
- etc...

R. Collobert, S. Bengio, and J. Mariéthoz, “Torch: a modular machine learning software library,” Idiap-RR Idiap-RR-46-2002, IDIAP, 2002.

A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, “Automatic differentiation in machine learning: a survey,” *Journal of machine learning research*, vol. 18, no. 153, 2018.

C. H. Bischof, A. Carle, P. Khademi, and A. Mauer, “ADIFOR 2.0: Automatic differentiation of Fortran 77 programs,” *IEEE Computational Science & Engineering*, vol. 3, no. 3, pp. 18–32, 1996.

C. Elliott, “The simple essence of automatic differentiation,” *Proceedings of the ACM on Programming Languages*, vol. 2, no. ICFP, p. 70, 2018.

A. Griewank and A. Walther, *Evaluating derivatives: principles and techniques of algorithmic differentiation*, vol. 105. Siam, 2008.

J. L. Elman, “Finding structure in time,” *Cognitive science*, vol. 14, no. 2, pp. 179–211, 1990.

L. Hascoët and V. Pascual, “The Tapenade automatic differentiation tool: Principles, model, and specification,” *ACM Transactions on Mathematical Software*, vol. 39, no. 3, pp. 20:1–20:43, 2013.

Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, “Efficient backprop,” in *Neural networks: Tricks of the trade*, pp. 9–48, Springer, 2012.

R. Alexander, “Solving ordinary differential equations i: Nonstiff problems (e. hairer, sp norsett, and g. wanner),” *Siam Review*, vol. 32, no. 3, p. 485, 1990.

U. Naumann, “Optimal jacobian accumulation is np-complete,” *Mathematical Programming*, vol. 112, no. 2, pp. 427–441, 2008.

R. Rojas, *Neural networks: a systematic introduction*. Springer Science & Business Media, 1996.