

Differentiable causal computations via delayed trace

Shin-ya Katsumata and David Sprunger*
National Institute of Informatics, Japan
ERATO MMSD

LICS 2019
SFU, Vancouver, CA
June 24, 2019



Machine learning has made incredible progress. . .



Image: Yonhap/Reuters

Machine learning has made incredible progress. . .



Image: Yonhap/Reuters

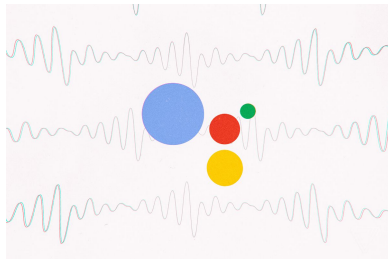


Image: Alex Castro

Machine learning has made incredible progress...



Image: Yonhap/Reuters

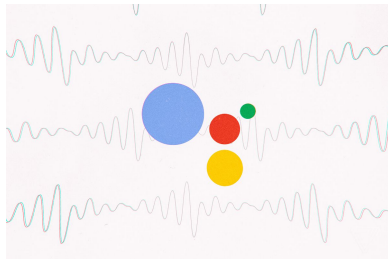


Image: Alex Castro



Image: Kundu, Vineet, Koltun

... but still faces incredible challenges.



Adversarial attacks

... but still faces incredible challenges.



Adversarial attacks

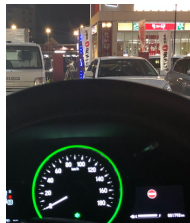


Real-world/training mismatch

... but still faces incredible challenges.



Adversarial attacks



Real-world/training mismatch



- Why did you do that?
- Why not something else?
- When do you succeed?
- When do you fail?
- When can I trust you?
- How do I correct an error?

Explainability

... but still faces incredible challenges.



Adversarial attacks



- Why did you do that?
- Why not something else?
- When do you succeed?
- When do you fail?
- When can I trust you?
- How do I correct an error?

Explainability

Real-world/training mismatch



Privacy

... but still faces incredible challenges.



Adversarial attacks



- Why did you do that?
- Why not something else?
- When do you succeed?
- When do you fail?
- When can I trust you?
- How do I correct an error?

Explainability

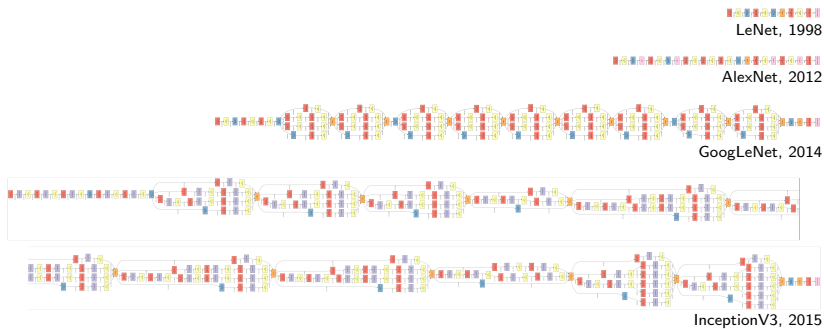
Real-world/training mismatch



Privacy

...and more!

Categorical tools may help



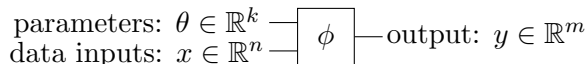
Images: Joseph Paul Cohen

Outline

- 1 Feedforward and recurrent neural networks
- 2 Statefulness, categorically
- 3 Differentiation, categorically
- 4 Training recurrent neural networks, categorically

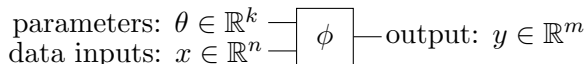
Feedforward neural networks

A *neural network* is a function taking *data inputs* (from the environment) and *parameters* (from us). Diagrammatically:

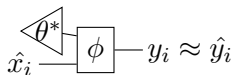


Feedforward neural networks

A *neural network* is a function taking *data inputs* (from the environment) and *parameters* (from us). Diagrammatically:

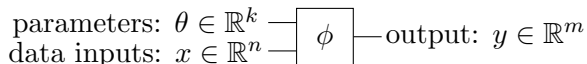


Training a neural network means finding $\theta^* : 1 \rightarrow \mathbb{R}^k$ so that

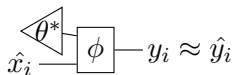


Feedforward neural networks

A *neural network* is a function taking *data inputs* (from the environment) and *parameters* (from us). Diagrammatically:



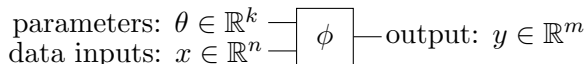
Training a neural network means finding $\theta^* : 1 \rightarrow \mathbb{R}^k$ so that



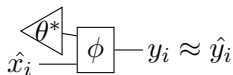
Gradient-based training algorithms are based on the insight that $\frac{\partial \phi}{\partial \theta}$ is a good approximation for the change in y that results from a small change in θ . This allows us to make smart updates to θ^* .

Feedforward neural networks

A *neural network* is a function taking *data inputs* (from the environment) and *parameters* (from us). Diagrammatically:



Training a neural network means finding $\theta^* : 1 \rightarrow \mathbb{R}^k$ so that

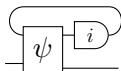


Gradient-based training algorithms are based on the insight that $\frac{\partial \phi}{\partial \theta}$ is a good approximation for the change in y that results from a small change in θ . This allows us to make smart updates to θ^* .

Backpropagation (Rumelhart '86) is an algorithm that finds derivatives of functions $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$, with excellent performance when $n \gg m$.

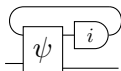
Recurrent neural networks

Recurrent neural networks (RNNs) process lists of inputs using *state*, which is stored in *registers*:

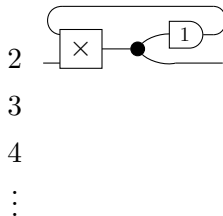


Recurrent neural networks

Recurrent neural networks (RNNs) process lists of inputs using *state*, which is stored in *registers*:

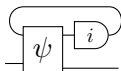


Operationally, these work as you expect:

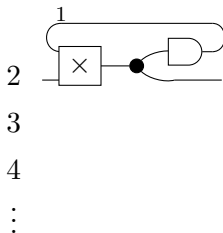


Recurrent neural networks

Recurrent neural networks (RNNs) process lists of inputs using *state*, which is stored in *registers*:

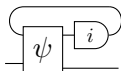


Operationally, these work as you expect:

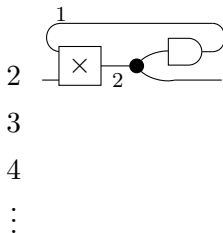


Recurrent neural networks

Recurrent neural networks (RNNs) process lists of inputs using *state*, which is stored in *registers*:

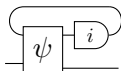


Operationally, these work as you expect:

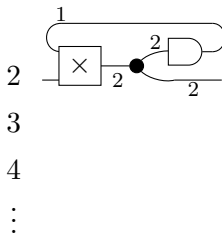


Recurrent neural networks

Recurrent neural networks (RNNs) process lists of inputs using *state*, which is stored in *registers*:

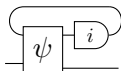


Operationally, these work as you expect:

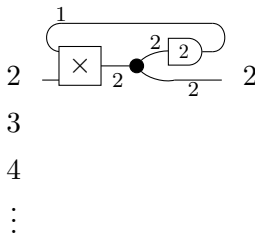


Recurrent neural networks

Recurrent neural networks (RNNs) process lists of inputs using *state*, which is stored in *registers*:

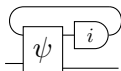


Operationally, these work as you expect:

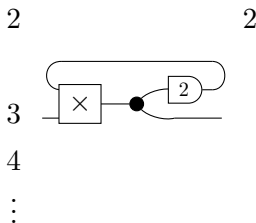


Recurrent neural networks

Recurrent neural networks (RNNs) process lists of inputs using *state*, which is stored in *registers*:

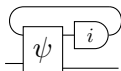


Operationally, these work as you expect:

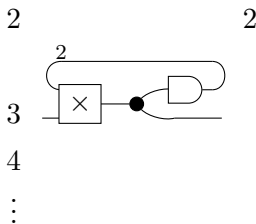


Recurrent neural networks

Recurrent neural networks (RNNs) process lists of inputs using *state*, which is stored in *registers*:

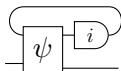


Operationally, these work as you expect:

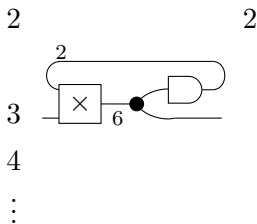


Recurrent neural networks

Recurrent neural networks (RNNs) process lists of inputs using *state*, which is stored in *registers*:

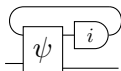


Operationally, these work as you expect:

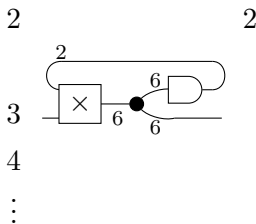


Recurrent neural networks

Recurrent neural networks (RNNs) process lists of inputs using *state*, which is stored in *registers*:

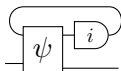


Operationally, these work as you expect:

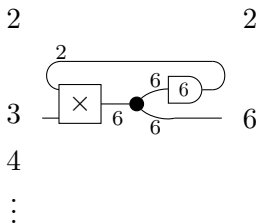


Recurrent neural networks

Recurrent neural networks (RNNs) process lists of inputs using *state*, which is stored in *registers*:

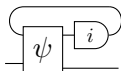


Operationally, these work as you expect:

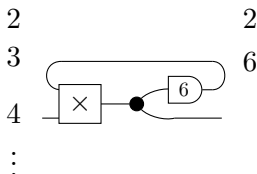


Recurrent neural networks

Recurrent neural networks (RNNs) process lists of inputs using *state*, which is stored in *registers*:

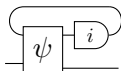


Operationally, these work as you expect:

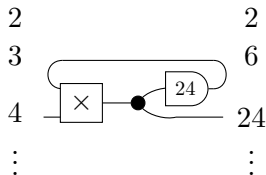


Recurrent neural networks

Recurrent neural networks (RNNs) process lists of inputs using *state*, which is stored in *registers*:



Operationally, these work as you expect:

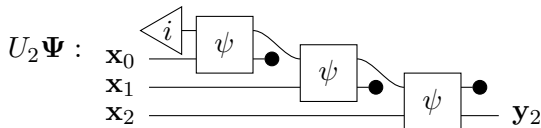
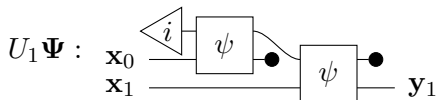
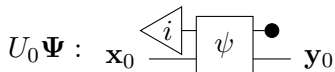


Unrollings and backpropagation through time (BPTT)

How do we train an RNN?

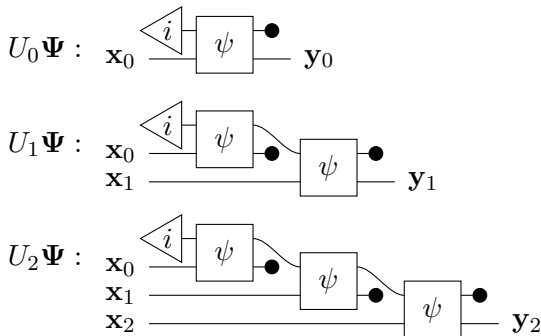
Unrollings and backpropagation through time (BPTT)

How do we train an RNN? Usually, we use its *unrollings*:



Unrollings and backpropagation through time (BPTT)

How do we train an RNN? Usually, we use its *unrollings*:



Backpropagation through time (Werbos '90): Whenever the derivative of Ψ is needed at an input of length $k + 1$, the derivative of $U_k\Psi$ is used instead.

Key questions

BPTT is useful in practice and makes sense with these operational semantics.

Key questions

BPTT is useful in practice and makes sense with these operational semantics. However, some basic theoretical properties went unaddressed in the literature:

- ① $U_k(\Psi \circ \Phi) \neq U_k \Psi \circ U_k \Phi$. Can we save the chain rule? What properties of derivatives hold for BPTT?

Key questions

BPTT is useful in practice and makes sense with these operational semantics. However, some basic theoretical properties went unaddressed in the literature:

- 1 $U_k(\Psi \circ \Phi) \neq U_k\Psi \circ U_k\Phi$. Can we save the chain rule? What properties of derivatives hold for BPTT?
- 2 $U_k\Psi$ and $U_{k+1}\Psi$ have a lot in common; can we use this fact to get a more compact representation for the derivative of Ψ ?

Key questions

BPTT is useful in practice and makes sense with these operational semantics. However, some basic theoretical properties went unaddressed in the literature:

- 1 $U_k(\Psi \circ \Phi) \neq U_k\Psi \circ U_k\Phi$. Can we save the chain rule? What properties of derivatives hold for BPTT?
- 2 $U_k\Psi$ and $U_{k+1}\Psi$ have a lot in common; can we use this fact to get a more compact representation for the derivative of Ψ ?

Key questions

BPTT is useful in practice and makes sense with these operational semantics. However, some basic theoretical properties went unaddressed in the literature:

- 1 $U_k(\Psi \circ \Phi) \neq U_k \Psi \circ U_k \Phi$. Can we save the chain rule? What properties of derivatives hold for BPTT?
- 2 $U_k \Psi$ and $U_{k+1} \Psi$ have a lot in common; can we use this fact to get a more compact representation for the derivative of Ψ ?

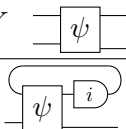
Besides these, is it possible to incorporate differentiation directly within a categorical framework?

Outline

- 1 Feedforward and recurrent neural networks
- 2 Statefulness, categorically**
- 3 Differentiation, categorically
- 4 Training recurrent neural networks, categorically

Delayed trace

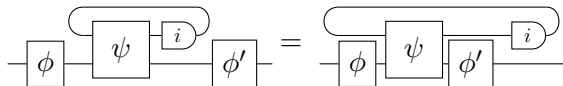
This loop-with-register is a like a trace (Joyal+ '96)—we call it a *delayed trace*, meaning it satisfies the trace axioms except yanking and dinaturality.

$$\frac{\psi : S \times X \rightarrow S \times Y}{dtr_i^S(\psi) : X \rightarrow Y}$$


Delayed trace

This loop-with-register is a like a trace (Joyal+ '96)—we call it a *delayed trace*, meaning it satisfies the trace axioms except yanking and dinaturality.

S/T naturality:



$$\frac{\psi : S \times X \rightarrow S \times Y}{dtr_i^S(\psi) : X \rightarrow Y}$$

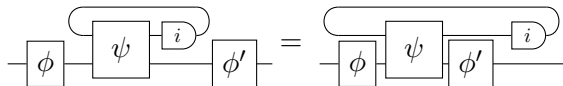
The diagram shows the definition of the delayed trace operation $dtr_i^S(\psi)$. It consists of a box labeled ψ with a loop labeled i on its right side, connected to another box labeled ψ with a loop labeled i on its right side.

Delayed trace

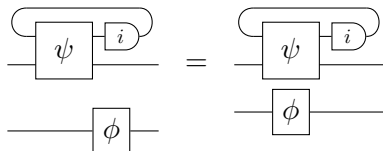
This loop-with-register is a like a trace (Joyal+ '96)—we call it a *delayed trace*, meaning it satisfies the trace axioms except yanking and dinaturality.

$$\frac{\psi : S \times X \rightarrow S \times Y}{dtr_i^S(\psi) : X \rightarrow Y}$$

S/T naturality:



Superposition:

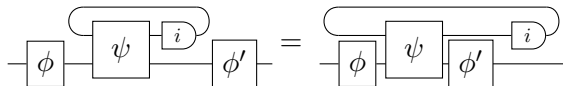


Delayed trace

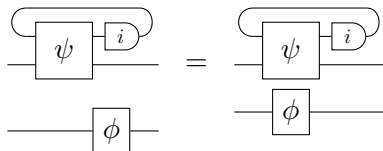
This loop-with-register is a like a trace (Joyal+ '96)—we call it a *delayed trace*, meaning it satisfies the trace axioms except yanking and dinaturality.

$$\frac{\psi : S \times X \rightarrow S \times Y}{dtr_i^S(\psi) : X \rightarrow Y}$$

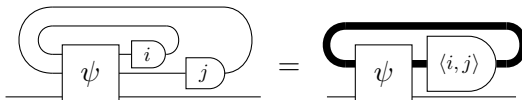
S/T naturality:



Superposition:



Vanishing \times :



$\text{St}(\mathbb{C})$ construction

We give a construction extending a Cartesian category \mathbb{C} to another $\text{St}(\mathbb{C})$ with a designated delayed trace, which models the operational semantics for RNNs.

$\text{St}(\mathbb{C})$ construction

We give a construction extending a Cartesian category \mathbb{C} to another $\text{St}(\mathbb{C})$ with a designated delayed trace, which models the operational semantics for RNNs. (Think of \mathbb{C} as feedforward networks and $\text{St}(\mathbb{C})$ as recurrent networks.)

$\text{St}(\mathbb{C})$ construction

We give a construction extending a Cartesian category \mathbb{C} to another $\text{St}(\mathbb{C})$ with a designated delayed trace, which models the operational semantics for RNNs. (Think of \mathbb{C} as feedforward networks and $\text{St}(\mathbb{C})$ as recurrent networks.)

Proposition: Normal forms for $\text{St}(\mathbb{C})$

Every $\Psi \in \text{St}(\mathbb{C})(\mathbf{A}, \mathbf{B})$ can be written as $\Psi = \text{dtr}_i^{\mathbf{S}}(\psi)$ for some $\psi \in \mathbb{C}(S \times A, S \times B)$ and $i : 1 \rightarrow S$.

$\text{St}(\mathbb{C})$ construction

We give a construction extending a Cartesian category \mathbb{C} to another $\text{St}(\mathbb{C})$ with a designated delayed trace, which models the operational semantics for RNNs. (Think of \mathbb{C} as feedforward networks and $\text{St}(\mathbb{C})$ as recurrent networks.)

Proposition: Normal forms for $\text{St}(\mathbb{C})$

Every $\Psi \in \text{St}(\mathbb{C})(\mathbf{A}, \mathbf{B})$ can be written as $\Psi = \text{dtr}_i^S(\psi)$ for some $\psi \in \mathbb{C}(S \times A, S \times B)$ and $i : 1 \rightarrow S$.

Adding state to computations is common:

- 1 Bicategorically—Katis, Sabadini, & Walters '97
- 2 Digital circuits—Ghica & Jung, '16
- 3 Signal flow graphs—Bonchi, Sobociński, & Zanasi, '14

Outline

- 1 Feedforward and recurrent neural networks
- 2 Statefulness, categorically
- 3 Differentiation, categorically
- 4 Training recurrent neural networks, categorically

Cartesian differential categories [Blute, Cockett, Seely '09]

Cartesian differential categories are the tool we use to introduce differentiation into our categorical framework.

Cartesian differential categories [Blute, Cockett, Seely '09]

Cartesian differential categories are the tool we use to introduce differentiation into our categorical framework. In detail, \mathbb{C} is a Cartesian differential category means:

- 1 \mathbb{C} is Cartesian (has \times and a terminal object 1),

Cartesian differential categories [Blute, Cockett, Seely '09]

Cartesian differential categories are the tool we use to introduce differentiation into our categorical framework. In detail, \mathbb{C} is a Cartesian differential category means:

- ① \mathbb{C} is Cartesian (has \times and a terminal object 1),
- ② every object has a chosen commutative monoid structure, meaning there are $0_X : 1 \rightarrow X$ and $+_X : X \times X \rightarrow X$ satisfying:
 - $+_X \circ (0_X \times \text{id}_X) = \text{id}_X$
 - $+_X \circ \sigma_{X,X} = +_X$,

Cartesian differential categories [Blute, Cockett, Seely '09]

Cartesian differential categories are the tool we use to introduce differentiation into our categorical framework. In detail, \mathbb{C} is a Cartesian differential category means:

- ① \mathbb{C} is Cartesian (has \times and a terminal object 1),
- ② every object has a chosen commutative monoid structure, meaning there are $0_X : 1 \rightarrow X$ and $+_X : X \times X \rightarrow X$ satisfying:
 - $+_X \circ (0_X \times \text{id}_X) = \text{id}_X$
 - $+_X \circ \sigma_{X,X} = +_X$,
- ③ this commutative monoid structure interacts nicely with the Cartesian structure:
 - $0_{X \times Y} = 0_X \times 0_Y$
 - $+_{X \times Y} = (+_X \times +_Y) \circ (\text{id}_X \times \sigma_{Y,X} \times \text{id}_Y)$, and

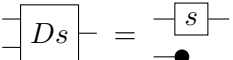
Cartesian differential categories [Blute, Cockett, Seely '09]

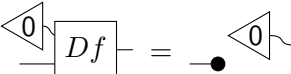
Cartesian differential categories are the tool we use to introduce differentiation into our categorical framework. In detail, \mathbb{C} is a Cartesian differential category means:

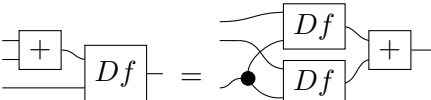
- ① \mathbb{C} is Cartesian (has \times and a terminal object 1),
- ② every object has a chosen commutative monoid structure, meaning there are $0_X : 1 \rightarrow X$ and $+_X : X \times X \rightarrow X$ satisfying:
 - $+_X \circ (0_X \times \text{id}_X) = \text{id}_X$
 - $+_X \circ \sigma_{X,X} = +_X$,
- ③ this commutative monoid structure interacts nicely with the Cartesian structure:
 - $0_{X \times Y} = 0_X \times 0_Y$
 - $+_{X \times Y} = (+_X \times +_Y) \circ (\text{id}_X \times \sigma_{Y,X} \times \text{id}_Y)$, and
- ④ there is a *Cartesian differential operator* on morphisms sending $f : X \rightarrow Y$ to $Df : X \times X \rightarrow Y$ satisfying seven axioms...

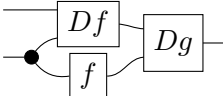
Cartesian differential operator axioms

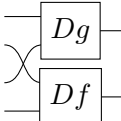
A Cartesian differential operator sending $f : X \rightarrow Y$ to $Df : X \times X \rightarrow Y$ satisfies these axioms:

CD1.  for $s \in \{\text{id}_X, \sigma_{X,Y}, !_X, \Delta_X, 0_X, +_X\}$.

CD2. 

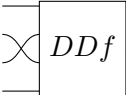
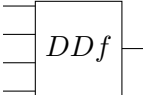
CD3. 

CD4. $D(-\boxed{f}-\boxed{g}-) =$ 

CD5. $D(\begin{smallmatrix} \boxed{g} \\ \boxed{f} \end{smallmatrix}) =$ 

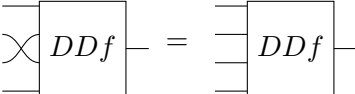
Cartesian differential operator axioms, continued

CD6.  = 

CD7.  = 

Cartesian differential operator axioms, continued

CD6. 



CD7. 

Example

Objects of the category \mathbf{Euc}_∞ are \mathbb{R}^n for $n \in \mathbb{N}$, maps are smooth functions between them. \mathbf{Euc}_∞ is a Cartesian differential category with the (curried) Jacobian sending $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ to $Df : (\Delta x, x) \mapsto Jf|_x \times \Delta x$.

Cartesian differential operator axioms, continued

CD6.  = 

CD7.  = 

Example

Objects of the category \mathbf{Euc}_∞ are \mathbb{R}^n for $n \in \mathbb{N}$, maps are smooth functions between them. \mathbf{Euc}_∞ is a Cartesian differential category with the (curried) Jacobian sending $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ to $Df : (\Delta x, x) \mapsto Jf|_x \times \Delta x$.

Suppose \mathbb{C} is a Cartesian differential category. Can we give $\mathbf{St}(\mathbb{C})$ a differential operator as well?

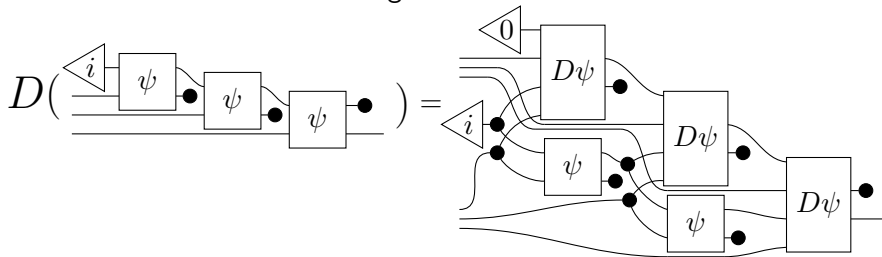
Differentiating RNN unrollings

$$D(\triangleleft i \mid \psi \bullet) = \begin{array}{c} \triangleleft 0 \\ \triangleleft i \end{array} \mid D\psi \bullet$$

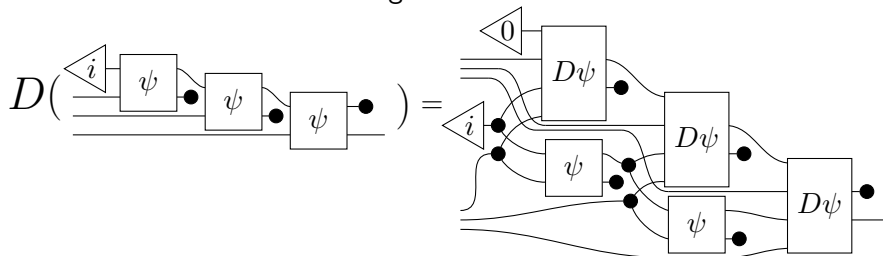
$$D(\triangleleft i \mid \psi \bullet \mid \psi \bullet) = \begin{array}{c} \triangleleft 0 \\ \triangleleft i \end{array} \mid \begin{array}{c} D\psi \\ \psi \end{array} \bullet \mid D\psi \bullet$$

$$D(\triangleleft i \mid \psi \bullet \mid \psi \bullet \mid \psi \bullet) = \begin{array}{c} \triangleleft 0 \\ \triangleleft i \end{array} \mid \begin{array}{c} D\psi \\ \psi \\ D\psi \\ \psi \end{array} \bullet \mid \begin{array}{c} D\psi \\ \psi \end{array} \bullet \mid D\psi \bullet$$

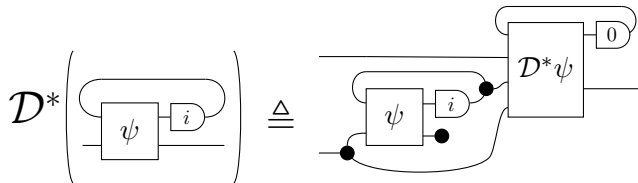
What could this be the unrolling of?



What could this be the unrolling of of?



Enough staring suggests the *recurrence rule*:



Main contributions

Theorem: causal differentiation

If \mathbb{C} is a Cartesian differential category with differential operator D , then $\text{St}(\mathbb{C})$ is also Cartesian differential with a related operator \mathcal{D}^* .

(The differential operator \mathcal{D}^* is basically D augmented with the recurrence rule.)

Main contributions

Theorem: causal differentiation

If \mathbb{C} is a Cartesian differential category with differential operator D , then $\text{St}(\mathbb{C})$ is also Cartesian differential with a related operator \mathcal{D}^* .

(The differential operator \mathcal{D}^* is basically D augmented with the recurrence rule.)

Theorem: categorical BPTT

$DU_k(\Psi) = U_k(\mathcal{D}^*\Psi) \circ z_k$ for all $k \in \mathbb{N}$ and $\Psi \in \text{St}(\mathbb{C})(\mathbf{A}, \mathbf{B})$.
($z : (\prod X_i) \times (\prod Y_i) \rightarrow \prod (X_i \times Y_i)$ is a zipping isomorphism.)

Main contributions

Theorem: causal differentiation

If \mathbb{C} is a Cartesian differential category with differential operator D , then $\text{St}(\mathbb{C})$ is also Cartesian differential with a related operator \mathcal{D}^* .

(The differential operator \mathcal{D}^* is basically D augmented with the recurrence rule.)

Theorem: categorical BPTT

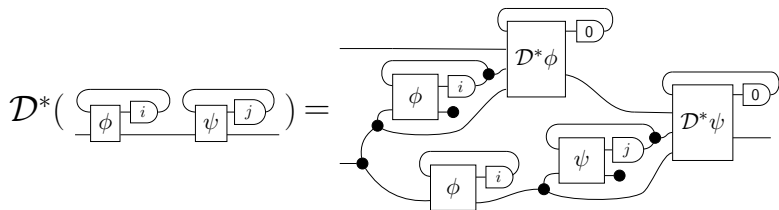
$DU_k(\Psi) = U_k(\mathcal{D}^*\Psi) \circ z_k$ for all $k \in \mathbb{N}$ and $\Psi \in \text{St}(\mathbb{C})(\mathbf{A}, \mathbf{B})$.
($z : (\prod X_i) \times (\prod Y_i) \rightarrow \prod (X_i \times Y_i)$ is a zipping isomorphism.)

Upshot: All the properties of derivatives in Cartesian differential categories hold for backpropagation through time!

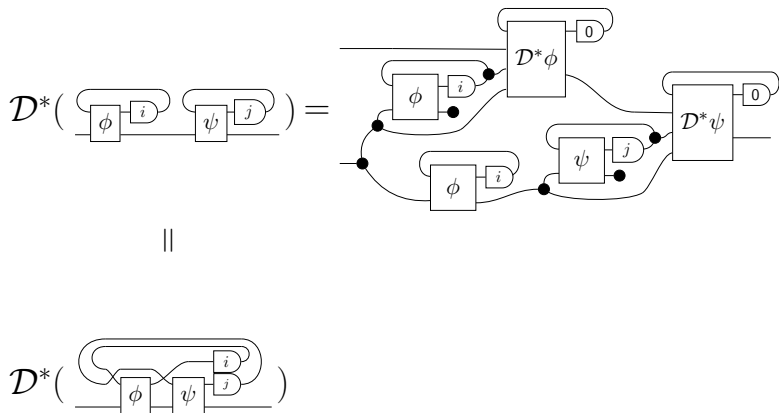
Example: stateful chain rule

$$\mathcal{D}^* \left(\begin{array}{c} \text{---} \boxed{\phi} \text{---} \boxed{i} \text{---} \boxed{\psi} \text{---} \boxed{j} \text{---} \\ \text{---} \end{array} \right)$$

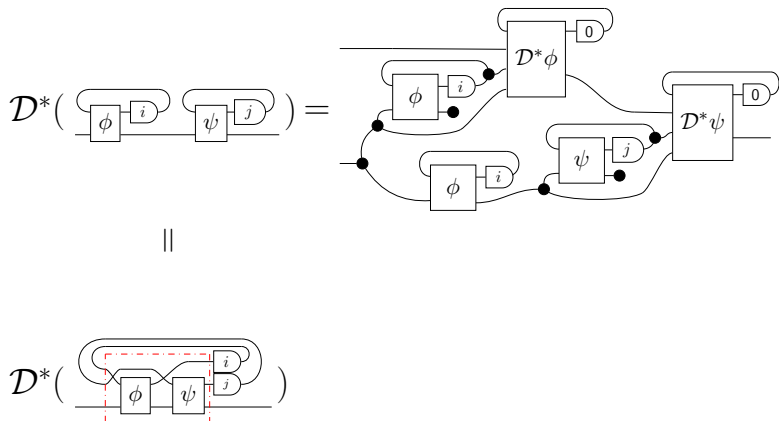
Example: stateful chain rule



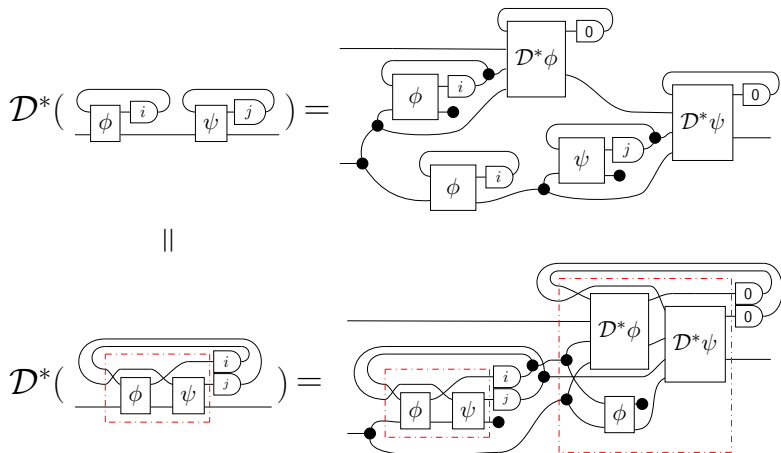
Example: stateful chain rule



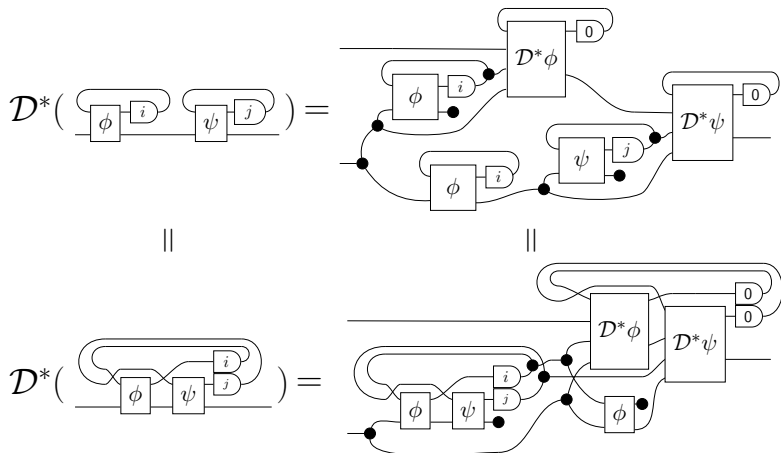
Example: stateful chain rule



Example: stateful chain rule



Example: stateful chain rule

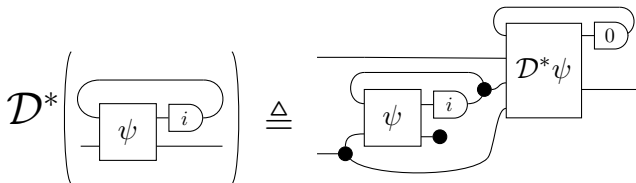


Messages

- 1 Given a Cartesian differential category \mathbb{C} , we can construct $\text{St}(\mathbb{C})$, another CDC with stateful functions (delayed trace).

Messages

- 1 Given a Cartesian differential category \mathbb{C} , we can construct $\text{St}(\mathbb{C})$, another CDC with stateful functions (delayed trace).
- 2 Differentiation for stateful functions includes a recurrence rule:



Messages

- 1 Given a Cartesian differential category \mathbb{C} , we can construct $\text{St}(\mathbb{C})$, another CDC with stateful functions (delayed trace).
- 2 Differentiation for stateful functions includes a recurrence rule:

$$\mathcal{D}^* \left(\psi \circ i \right) \triangleq \mathcal{D}^* \psi \circ i$$

- 3 Stateful differentiation is connected to BPTT, and we can find many useful properties of BPTT with this connection.

Messages

- 1 Given a Cartesian differential category \mathbb{C} , we can construct $\text{St}(\mathbb{C})$, another CDC with stateful functions (delayed trace).
- 2 Differentiation for stateful functions includes a recurrence rule:

$$\mathcal{D}^* \left(\begin{array}{c} \text{---} \text{---} \text{---} \\ \text{---} \text{---} \text{---} \end{array} \right) \equiv \begin{array}{c} \text{---} \text{---} \text{---} \\ \text{---} \text{---} \text{---} \end{array}$$

- 3 Stateful differentiation is connected to BPTT, and we can find many useful properties of BPTT with this connection.
- 4 Category theory and compositionality can be useful tools to organize and find structure in machine learning.

Future questions (roughly theoretical to practical)

- 1 Categorical properties of $\text{St}(-)$?
- 2 \mathcal{D}^* and infinite-dimensional derivatives?
- 3 Bisimulations and extensional equality?
- 4 Basic results for delayed trace categories?
- 5 Extra structure for reverse-mode AD?
- 6 Probabilistic/nondeterministic causal functions?
- 7 Partial functions with differential restriction categories?
- 8 Iterations I: RNN parameters in $\text{St}(\text{St}(\mathbb{C}))$?
- 9 Iterations II: RNN hyperparameters in $\text{St}(\text{St}(\text{St}(\mathbb{C})))$?
- 10 Measuring complexity with string diagrams?
- 11 What happens when $\phi \in \{LSTM, GRU, \dots\}$?
- 12 Implementation/plugin to neural net library?

References



R.F. Blute, J.R.B. Cockett, and R.A.G. Seely.
Cartesian differential categories.
Theory and Applications of Categories, 22(23):622–672, 2009.



F. Bonchi, P. Sobociński, and F. Zanasi.
A categorical semantics of signal flow graphs.
In *CONCUR 2014*, 2014.



C. Elliott.
The simple essence of automatic differentiation.
PACMPL, 2(ICFP):70:1–70:29, 2018.



B. Fong, D. Spivak, and R. Tuyéras.
Backprop as functor: A compositional perspective on supervised learning.
See arxiv.org/abs/1711.10455, 2017.



D.R. Ghica and A. Jung.
Categorical semantics of digital circuits.
FMCAD '16, Austin, TX, 2016.



P. Katis, N. Sabadini, and R.F.C. Walters.
Bicategories of processes.
Journal of Pure and Applied Algebra, 115(2):141–178, Feb 1997.



P.J. Werbos.
Backpropagation through time: what it does and how to do it.
Proceedings of the IEEE, 78(10):1550–1560, Oct 1990.

Thanks!