# Differential Categories, Recurrent Neural Networks, and Machine Learning

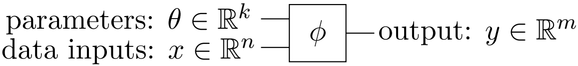
Shin-ya Katsumata and David Sprunger*
National Institute of Informatics, Tokyo

SYCO 4
Chapman University
May 23, 2019

# Outline

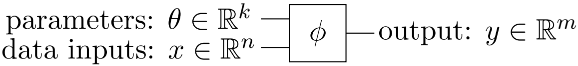
# Overview of neural networks

A *neural network* is a function with two types of arguments, *data inputs* and *parameters*. Data come from the environment, parameters are controlled by us. As a string diagram:

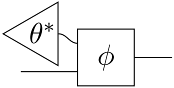


# Overview of neural networks

A *neural network* is a function with two types of arguments, *data inputs* and *parameters*. Data come from the environment, parameters are controlled by us. As a string diagram:



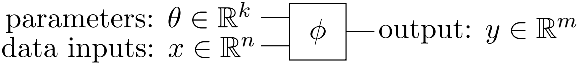

*Training* a neural network means finding $\theta^* : 1 \to \mathbb{R}^k$ so that



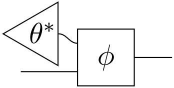

has a desired property.

# Overview of neural networks

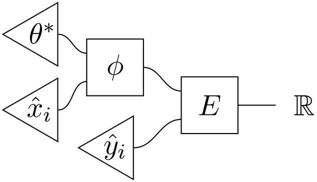
A *neural network* is a function with two types of arguments, *data inputs* and *parameters*. Data come from the environment, parameters are controlled by us. As a string diagram:




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



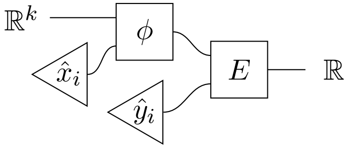

has a desired property. Usually, this means minimizing inaccuracy, as measured by




where $\langle \hat{x}_i, \hat{y}_i \rangle : 1 \to \mathbb{R}^{n+m}$ are given input-output pairs and $E : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$ is a given error function.

# Overview of neural networks

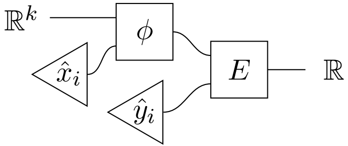
*Gradient-based training* algorithms utilize the insight that the gradient of this function:




tells us how to modify $\theta$ in order to decrease the error quickest.

# Overview of neural networks

*Gradient-based training* algorithms utilize the insight that the gradient of this function:

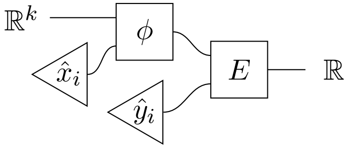



tells us how to modify $\theta$ in order to decrease the error quickest.

*Backpropagation* is an algorithm that finds gradients (or derivatives) of functions $f : \mathbb{R}^n \to \mathbb{R}^m$, and is often used due to its performance when $n \gg m$.

# Overview of neural networks

*Gradient-based training* algorithms utilize the insight that the gradient of this function:




tells us how to modify $\theta$ in order to decrease the error quickest.

*Backpropagation* is an algorithm that finds gradients (or derivatives) of functions $f : \mathbb{R}^n \to \mathbb{R}^m$, and is often used due to its performance when $n \gg m$.

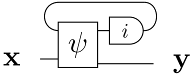
Backprop generates a hint about which direction to change $\theta$, but the trainer determines how this hint is used.

# Outline

# Recurrent neural networks

Recurrent neural networks (RNNs) are able to process variable-length inputs using *state*, which is stored in *registers*:

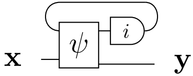
$\mathbf{\Psi}$ :



# Recurrent neural networks

Recurrent neural networks (RNNs) are able to process variable-length inputs using *state*, which is stored in *registers*:

$\mathbf{\Psi}$ :

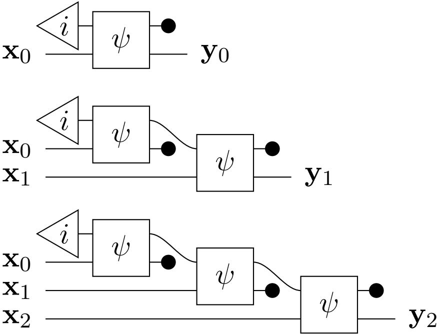



A common semantics of RNNs uses the *unrollings* of the network:

$U_0\mathbf{\Psi}$ :

$U_1\mathbf{\Psi}$ :

$U_2\mathbf{\Psi}$ :



# Backpropagation through time

Infinite-dimensional derivatives for sequence-to-sequence functions must be approximated to be computationally useful.

# Backpropagation through time

Infinite-dimensional derivatives for sequence-to-sequence functions must be approximated to be computationally useful.

*Backpropagation through time (BPTT)*: Whenever the derivative of $\mathbf{\Psi}$ is needed at an input of length $k+1$, the derivative of $U_k\mathbf{\Psi}$ is used instead.

# Backpropagation through time

Infinite-dimensional derivatives for sequence-to-sequence functions must be approximated to be computationally useful.

*Backpropagation through time (BPTT)*: Whenever the derivative of $\mathbf{\Psi}$ is needed at an input of length $k+1$, the derivative of $U_k\mathbf{\Psi}$ is used instead.

This is a good way to generate hints, but it opens some questions:

1. $U_k(\mathbf{\Psi} \circ \mathbf{\Phi}) \neq U_k\mathbf{\Psi} \circ U_k\mathbf{\Phi}$. Did we lose the chain rule? What properties of derivatives hold for BPTT?

# Backpropagation through time

Infinite-dimensional derivatives for sequence-to-sequence functions must be approximated to be computationally useful.

*Backpropagation through time (BPTT)*: Whenever the derivative of $\mathbf{\Psi}$ is needed at an input of length $k+1$, the derivative of $U_k\mathbf{\Psi}$ is used instead.

This is a good way to generate hints, but it opens some questions:

1. $U_k(\mathbf{\Psi} \circ \mathbf{\Phi}) \neq U_k\mathbf{\Psi} \circ U_k\mathbf{\Phi}$. Did we lose the chain rule? What properties of derivatives hold for BPTT?
2. $U_k\mathbf{\Psi}$ and $U_{k+1}\mathbf{\Psi}$ have a lot in common, so their derivatives should as well. Is there a more compact representation for the derivative of $\mathbf{\Psi}$ than a sequence of functions?

# Understanding BPTT with category theory

The project: start in a category with some notion of derivative, add a mechanism for state, and extend the original notion of differentiation to the stateful setting.

Two main parts:

1. Adding state to computations

2. Differentiation for stateful computations

# Understanding BPTT with category theory

The project: start in a category with some notion of derivative, add a mechanism for state, and extend the original notion of differentiation to the stateful setting.

Two main parts:

1. Adding state to computations
    1. Relatively common, back to Katis, Sabadini, & Walters '97
    2. Digital circuits—Ghica & Jung, '16
    3. Signal flow graphs—Bonchi, Sobociński, & Zanasi, '14
2. Differentiation for stateful computations

# Understanding BPTT with category theory

The project: start in a category with some notion of derivative, add a mechanism for state, and extend the original notion of differentiation to the stateful setting.

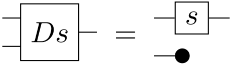
Two main parts:

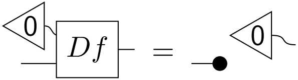
1. Adding state to computations
   1. Relatively common, back to Katis, Sabadini, & Walters ‘97
   2. Digital circuits—Ghica & Jung, ‘16
   3. Signal flow graphs—Bonchi, Sobociński, & Zanasi, ‘14
2. Differentiation for stateful computations
   1. Not so common
   2. Cartesian differential categories—Blute, Cockett, Seely ‘09
   3. (Backprop as Functor—Fong, Spivak, Tuyéras, ‘17)
   4. (Simple Essence of Automatic Differentiation—Elliott ‘18)

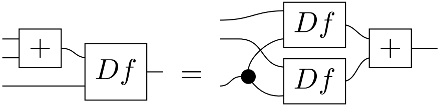
# Outline

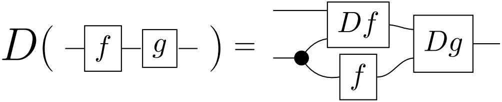
# Cartesian differential categories [Blute, Cockett, Seely '09]

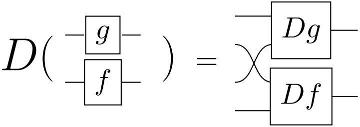
A *Cartesian differential category* has a differential operation on morphisms sending $f : X \to Y$ to $Df : X \times X \to Y$, satisfying seven axioms:

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

**CD2.** 

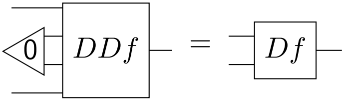
**CD3.** 

**CD4.** 

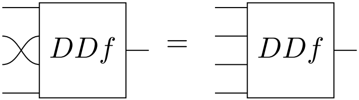
**CD5.** 

# Cartesian differential category axioms, continued

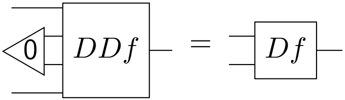
**CD6.**

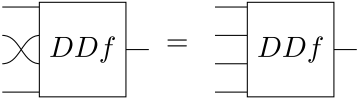



**CD7.**



# Cartesian differential category axioms, continued

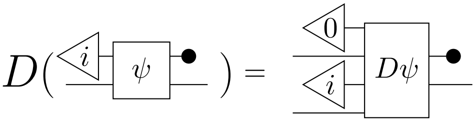
**CD6.** 


**CD7.** 

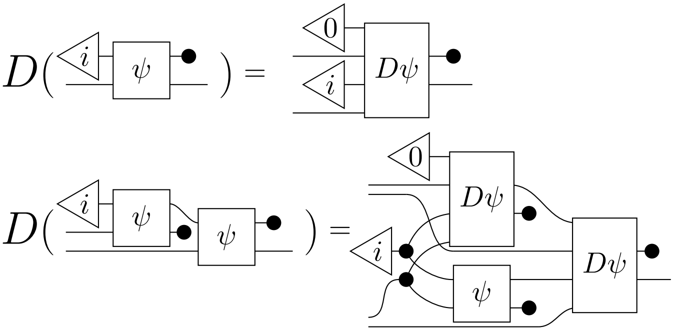

**Example**

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

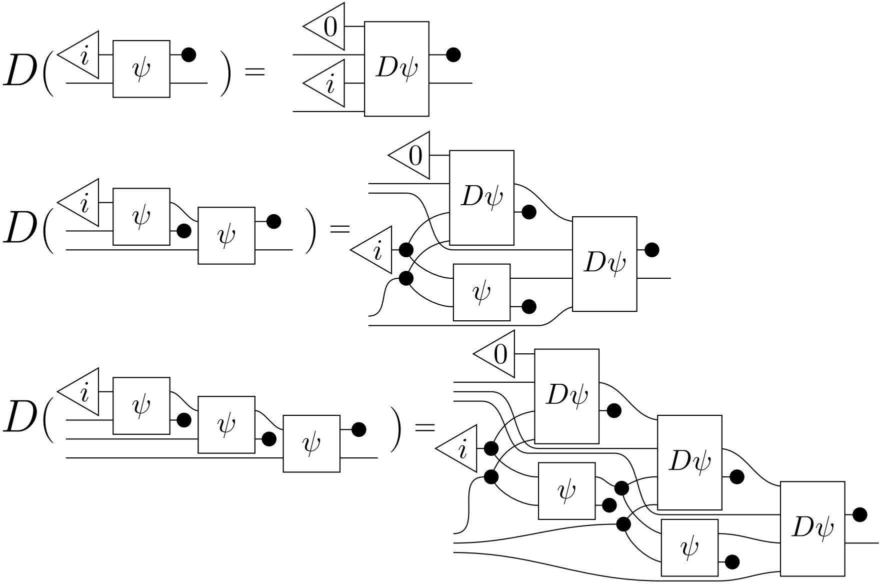
# Differentiating the unrollings of a simple RNN

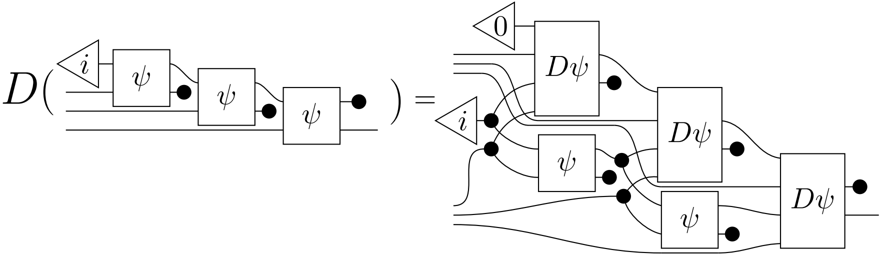


# Differentiating the unrollings of a simple RNN

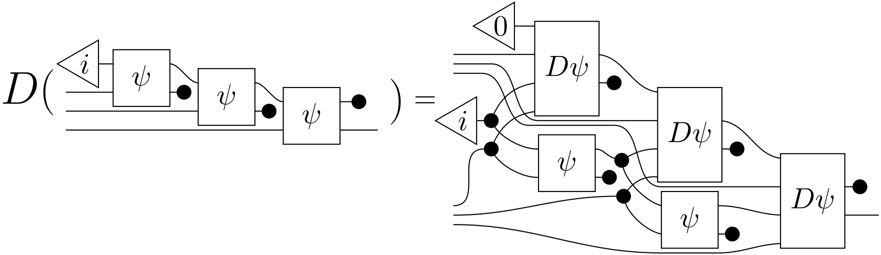


# Differentiating the unrollings of a simple RNN

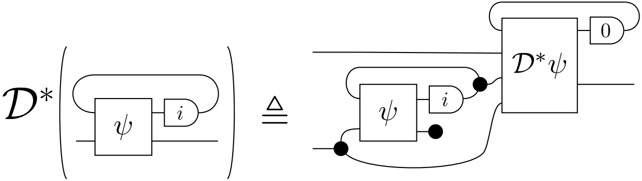





This suggests a hypothesis:



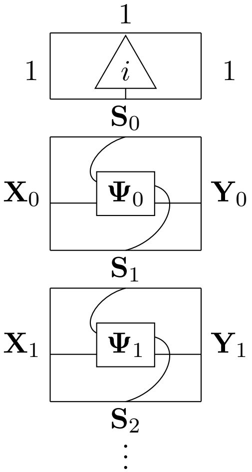

This suggests a hypothesis:



# Outline

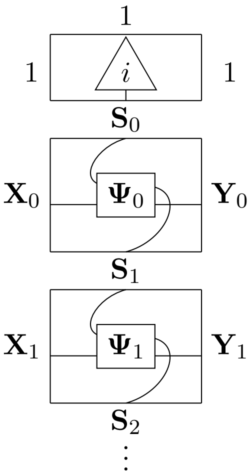
# Stateful **computations**

Let $(\mathbb{C}, \times, 1)$ be a strict Cartesian category, whose morphisms we think of as stateless functions. A stateful sequence computation looks like this:



# Stateful **computations**

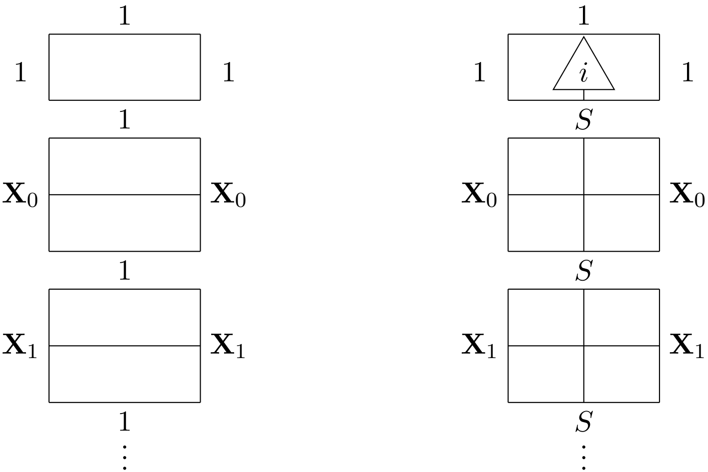
Let $(\mathbb{C}, \times, 1)$ be a strict Cartesian category, whose morphisms we think of as stateless functions. A stateful sequence computation looks like this:




(This is a sequence of 2-cells in a double category based on $\mathbb{C}$, with a restriction on the first 2-cell.)

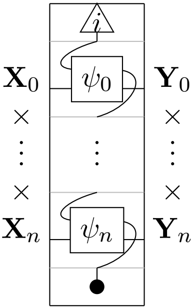
# Stateful **functions**

Two computation sequences might have different state spaces and still compute the same function. For example:



# Stateful **functions**

The $n$*th truncation* of a computation sequence is the morphism of the vertical composite of the first $n+1$ steps:




## Definition

Two computation sequences are *extensionally equivalent* means they have the same $n$th truncation for all $n \in \mathbb{N}$. A *stateful (sequence) function* is an extensional equivalence class of computation sequences.

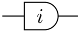
# Stateful **functions**

## Definition

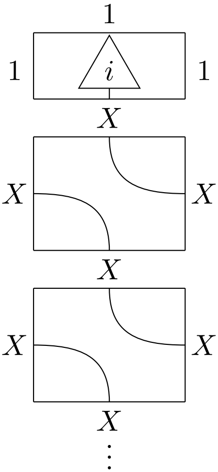
If $\mathbb{C}$ is a strict Cartesian category, then its *stateful sequence extension* is a category $\mathrm{St}(\mathbb{C})$ where

- objects are infinite sequences of objects in $\mathbb{C}$ and
- morphisms are stateful functions $\mathbf{\Psi} : \mathbf{X} \to \mathbf{Y}$.

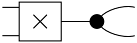
# Example computation sequences

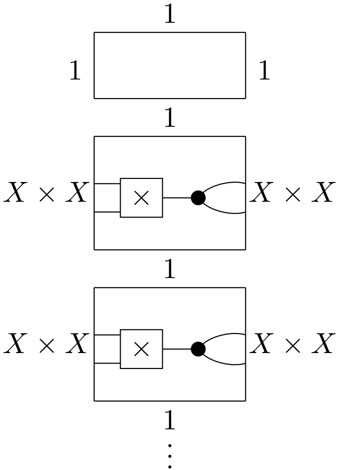
Here is  as a computation sequence:

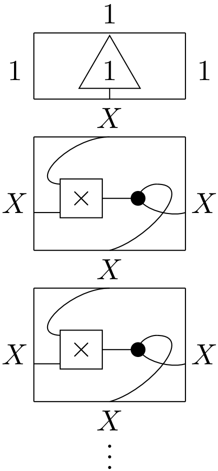




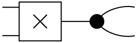
# Example computation sequences

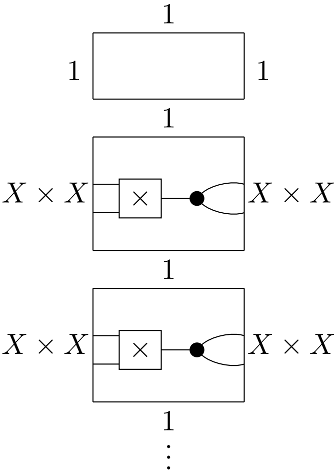
Here is  as a computation sequence:

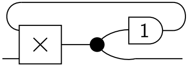


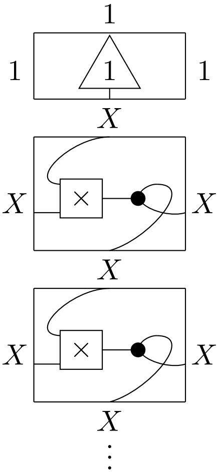
# Example computation sequences

Here is  as a computation sequence:




Here is  as a computation sequence:

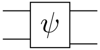


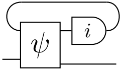


# Delayed trace

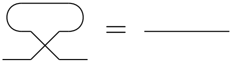
This loop-with-delay-gate is a trace-like operation.

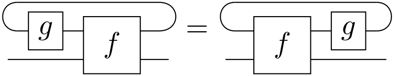
$$\frac{\psi : S \times X \to S \times Y}{dtr_i^S(\psi) : X \to Y}$$



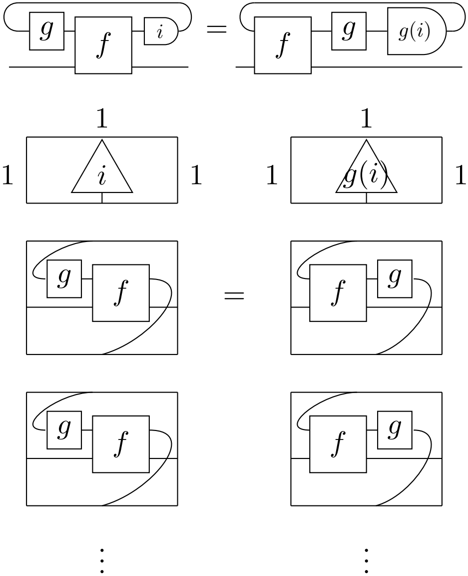


It satisfies **most** of the the trace axioms but misses two: yanking and dinaturality. For regular trace, those are

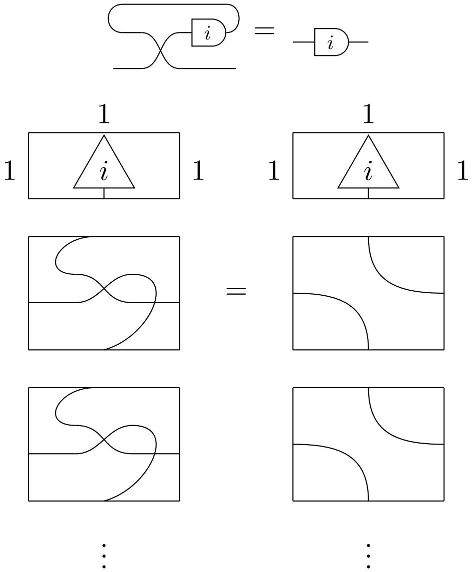




# ~~Dinaturality~~ $\to$ retiming



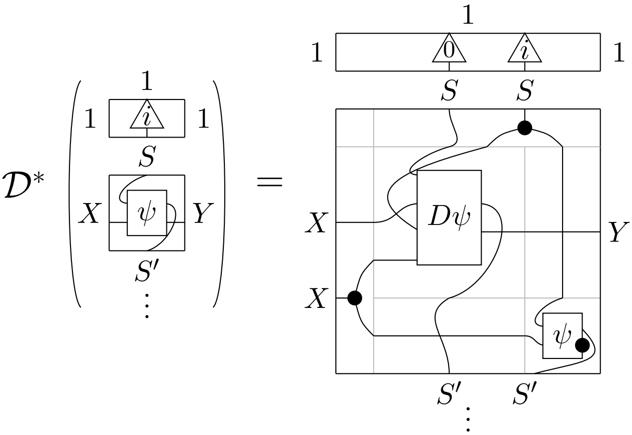
# ~~Yanking~~ $\to$ delay



# Outline

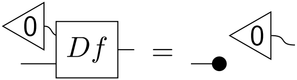
# Differentiation for stateful functions

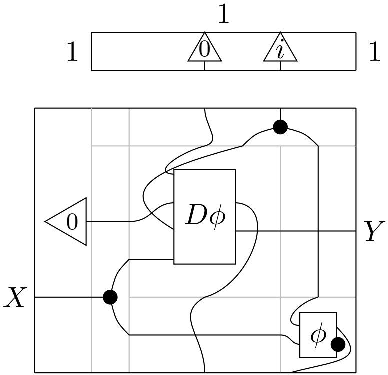
Let $\mathbb{C}$ be Cartesian differential with differential operator $D$. The following is a Cartesian differential operator on $\mathrm{St}(\mathbb{C})$:



# $\mathcal{D}^*$ is a Cartesian differential operator

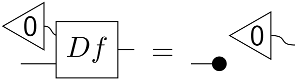
**Proof idea.** For CD2:

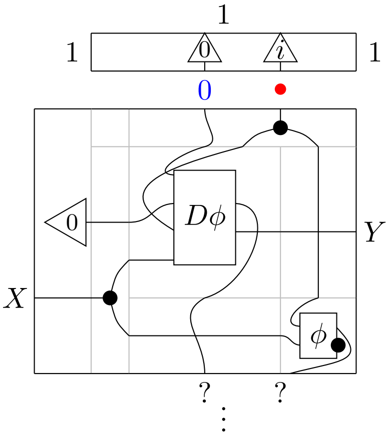


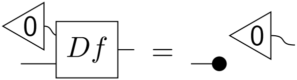

$\vdots$

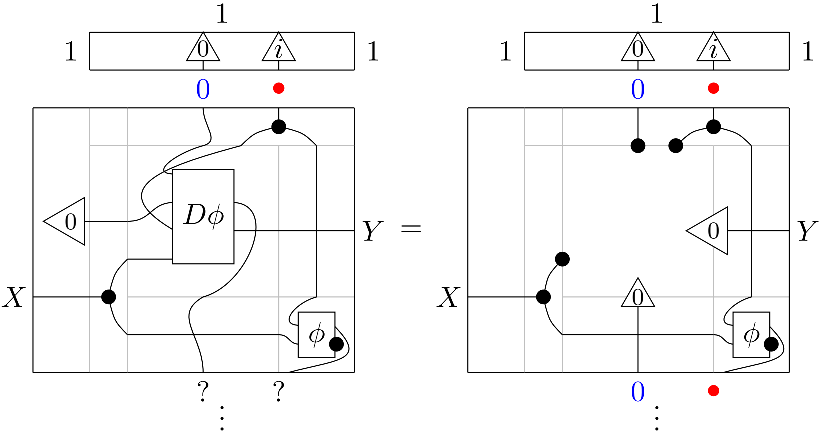
# $\mathcal{D}^*$ is a Cartesian differential operator

**Proof idea.** For CD2: 

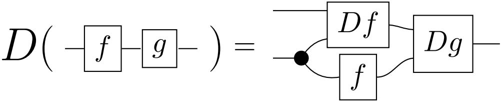
# $\mathcal{D}^*$ is a Cartesian differential operator

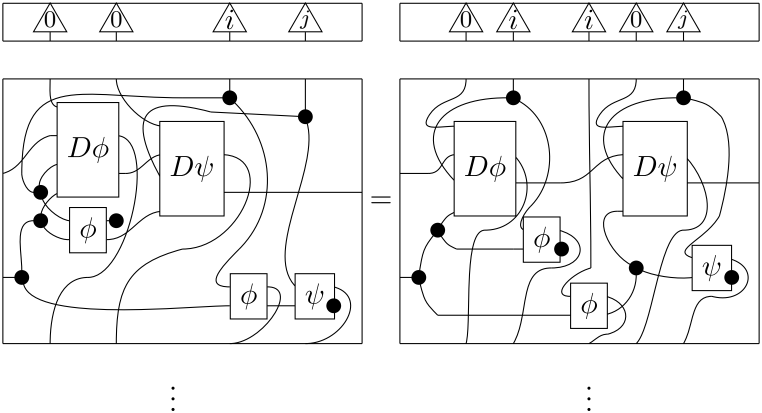
**Proof idea.** For CD2: 

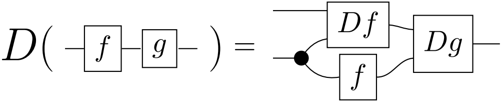
# $\mathcal{D}^*$ is a Cartesian differential operator

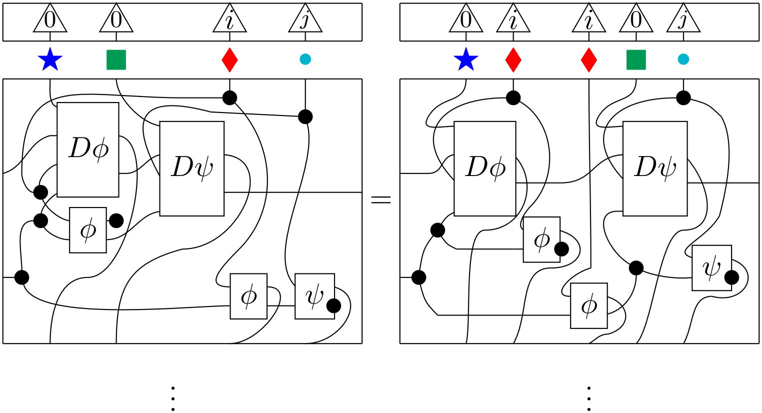
**Proof idea.** For CD4:



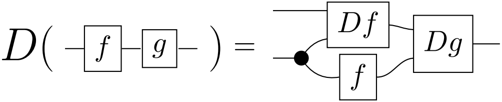
# $\mathcal{D}^*$ is a Cartesian differential operator

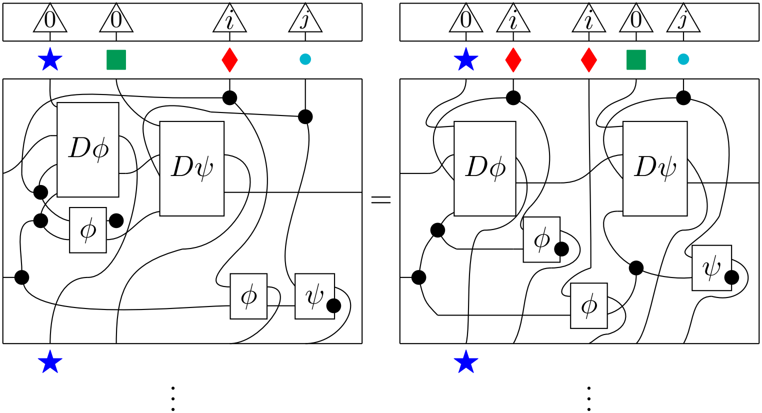
**Proof idea.** For CD4: 

# $\mathcal{D}^*$ is a Cartesian differential operator

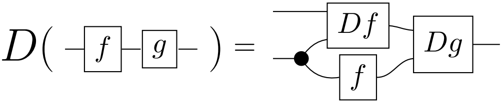
**Proof idea.** For CD4:

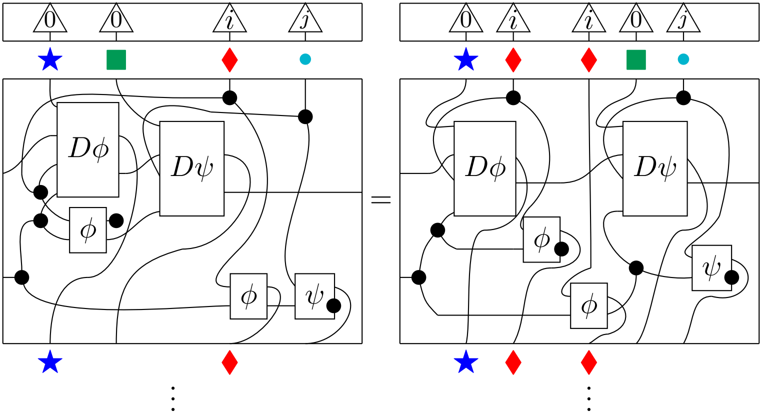


# $\mathcal{D}^*$ is a Cartesian differential operator

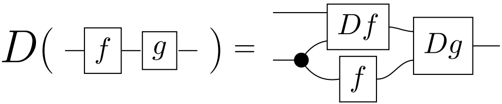
**Proof idea.** For CD4:

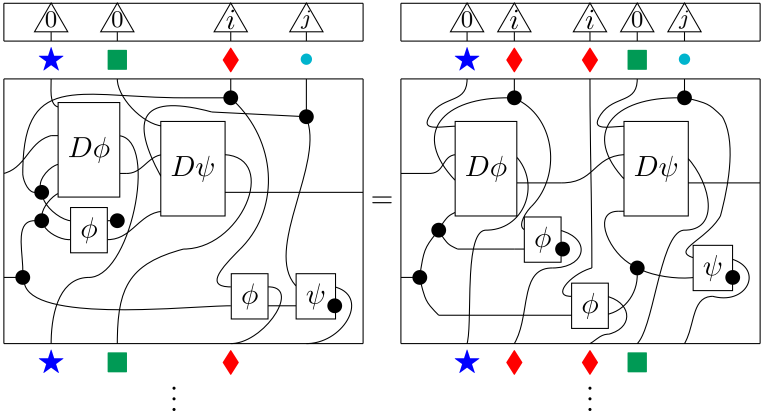


# $\mathcal{D}^*$ is a Cartesian differential operator

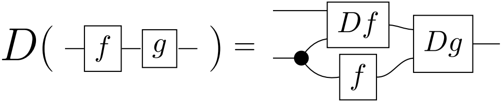
**Proof idea.** For CD4:

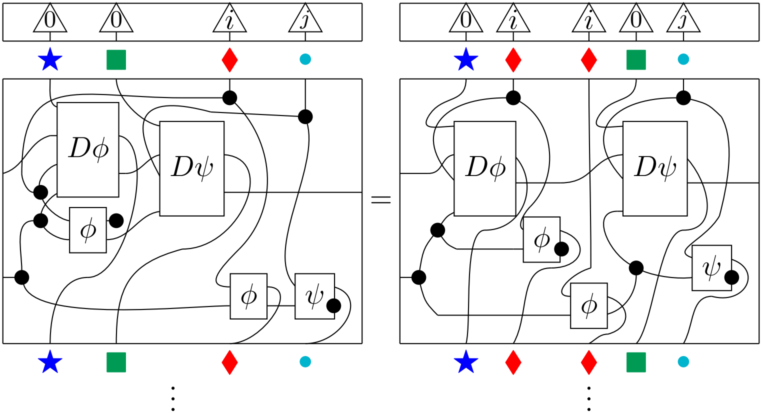


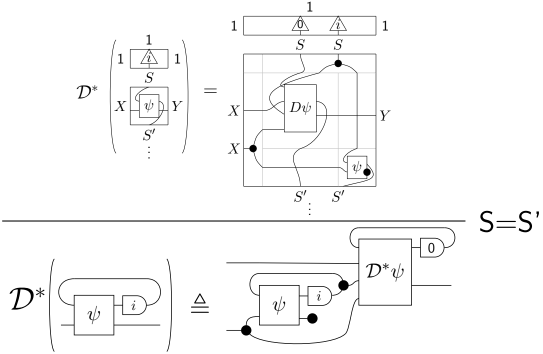
# $\mathcal{D}^*$ is a Cartesian differential operator

**Proof idea.** For CD4:

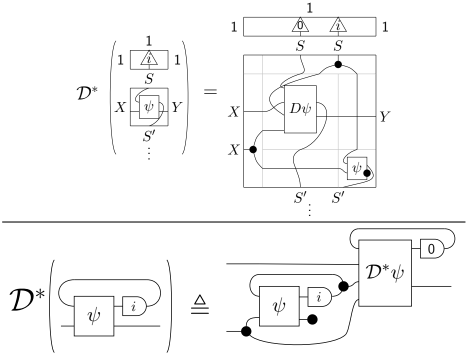


# Differentiating RNNs



# Differentiating RNNs




**Theorem ($\mathcal{D}^*$ matches BPTT)**

*The unrolling of* $\mathcal{D}^*(i, [\psi])$ *is the component-wise application of* $D$ *to the unrolling of* $(i, [\psi])$ *(after a zipping morphism).*

# Future directions

Several obstacles prevent us from applying these ideas in practice right away:

1. Use non-smooth, partly differentiable, or partial functions?
2. Can we get a transpose?
3. Better ways to represent non-mutable state?

# Future directions

Several obstacles prevent us from applying these ideas in practice right away:

1. Use non-smooth, partly differentiable, or partial functions?
2. Can we get a transpose?
3. Better ways to represent non-mutable state?

There are some questions related to the theory that we would like to understand better:

1. Categorical properties of $\mathrm{St}(-)$?
2. Bisimulations and extensional equality?
3. $\mathcal{D}^*$ and infinite-dimensional derivatives?
4. Basic results for delayed trace categories?
5. Other data shapes (trees, distributions, . . . )?

# Summary

1. $\mathrm{St}(-)$ preserves Cartesian differential category structure.
2. This notion of differentiation is connected to BPTT.

# Summary

1. $\mathrm{St}(-)$ preserves Cartesian differential category structure.
2. This notion of differentiation is connected to BPTT.
3. Cartesian differential categories are a useful tool for organizing unusual derivatives.

# Summary

1. $\mathrm{St}(-)$ preserves Cartesian differential category structure.
2. This notion of differentiation is connected to BPTT.
3. Cartesian differential categories are a useful tool for organizing unusual derivatives.
4. Machine learning needs compositional thinkers.

Thanks!

# 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.

David Sprunger and Shin-ya Katsumata.
Differentiable causal computations via delayed trace.
*CoRR*, abs/1903.01093, 2019.