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...
Machine learning has made incredible progress...
Machine learning has made incredible progress...
... 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

Explainability

- 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?
but still faces incredible challenges.

Adversarial attacks

Real-world/training mismatch

Explainability

Privacy
... but still faces incredible challenges.

Adversarial attacks

Real-world/training mismatch

Privacy

... and more!

Explainability
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:

parameters: \( \theta \in \mathbb{R}^k \)
data inputs: \( x \in \mathbb{R}^n \) \[ \phi \]
output: \( y \in \mathbb{R}^m \)
Feedforward neural networks

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

\[
\begin{align*}
\text{parameters: } & \theta \in \mathbb{R}^k \\
\text{data inputs: } & x \in \mathbb{R}^n \\
\phi & \quad \text{output: } y \in \mathbb{R}^m
\end{align*}
\]

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

\[
\hat{x}_i \quad \phi \quad y_i \approx \hat{y}_i
\]
Feedforward neural networks

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

\[
\begin{aligned}
\text{parameters: } \theta & \in \mathbb{R}^k \\
\text{data inputs: } x & \in \mathbb{R}^n \\
\phi & \quad \text{output: } y \in \mathbb{R}^m
\end{aligned}
\]

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

\[
\hat{x}_i \quad \phi \quad y_i \approx \hat{y}_i
\]

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 \( \theta \). This allows us to make smart updates to \( \theta^* \).
Feedforward neural networks

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

\[
\text{parameters: } \theta \in \mathbb{R}^k \quad \phi \quad \text{output: } y \in \mathbb{R}^m
\]

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

\[
\hat{x}_i \xrightarrow{\phi} y_i \approx \hat{y}_i
\]

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 \( \theta \). This allows us to make smart updates to \( \theta^* \).

Backpropagation (Rumelhart ‘86) is an algorithm that finds derivatives of functions \( f : \mathbb{R}^n \to \mathbb{R}^m \), 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*:

![Diagram of state and input in RNN](image-url)

where $\psi$ represents the state and $i$ the input.
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 (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:

\[ \psi \]

1

2

3

4

\[ \vdots \]
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:

\[
\begin{array}{c}
2 \\
\times \\
3 \\
4 \\
\vdots
\end{array}
\]
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 (RNNs) process lists of inputs using state, which is stored in registers:

\[ \psi \]

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:

```
  2   2
  3   6
  4   6
  ...```

Recurrent neural networks

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

```
ψ

Operationally, these work as you expect:
```

```
2
3
4
... 
```

```
2
6
```

```

× 
```

```
```

× 
```

6
Recurrent neural networks

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

![Diagram of RNN processing](image)

Operationally, these work as you expect:

\[
\begin{array}{c}
2 \\
3 \\
4 \\
\vdots \\
\end{array}
\quad \begin{array}{c}
2 \\
6 \\
24 \\
\vdots \\
\end{array}
\]
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:

\[
U_0 \Psi : \quad x_0 \psi \rightarrow y_0
\]

\[
U_1 \Psi : \quad x_0, x_1 \psi \rightarrow y_1
\]

\[
U_2 \Psi : \quad x_0, x_1, x_2 \psi \rightarrow y_2
\]
Unrollings and backpropagation through time (BPTT)

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

\[ U_0 \Psi : \quad x_0 \rightarrow \psi \rightarrow y_0 \]

\[ U_1 \Psi : \quad x_0, x_1 \rightarrow \psi \rightarrow \psi \rightarrow y_1 \]

\[ U_2 \Psi : \quad x_0, x_1, x_2 \rightarrow \psi \rightarrow \psi \rightarrow \psi \rightarrow y_2 \]

*Backpropagation through time* (Werbos ‘90): Whenever the derivative of \( \Psi \) 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 $\Psi$?
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 $\Psi$?
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 $\Psi$?

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.

\[
\psi : S \times X \to S \times Y
\]

\[
dtr^S_i(\psi) : X \to Y
\]
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:
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:

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.

\[ \psi : S \times X \rightarrow S \times Y \]

\[ dtr^S_i(\psi) : X \rightarrow Y \]

S/T naturality:

\[ \phi \psi i = \phi' \psi i \]

Superposition:

\[ \psi i = \psi \]

Vanishing \times:

\[ \psi i j = \psi \langle i, j \rangle \]
St($\mathcal{C}$) construction

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

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

Proposition: Normal forms for St(\mathcal{C})

Every \Psi \in St(\mathcal{C})(A, B) can be written as \Psi = dtr_i(\psi) for some \psi \in \mathcal{C}(S \times A, S \times B) and i: 1 \to S.
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 $\Psi \in \text{St}(\mathbb{C})(\mathbb{A}, \mathbb{B})$ can be written as $\Psi = dtr^S_i(\psi)$ for some $\psi \in \mathbb{C}(S \times \mathbb{A}, S \times \mathbb{B})$ and $i : 1 \to S$. 
**St(\mathcal{C}) construction**

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

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

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

Adding state to computations is common:

1. Bicategorically—Katis, Sabadini, & Walters ‘97
2. Digital circuits—Ghica & Jung, ‘16
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, \( C \) is a Cartesian differential category means:

1. \( C \) is Cartesian (has \( \times \) and a terminal object \( 1 \)),

\[
\begin{align*}
\forall X, Y \in C, & \quad 0_X \times 0_Y = 0_X \times 0_Y + X \times Y = (0_X \times +_Y) \circ (\text{id}_X \times \sigma_{Y,X} \times \text{id}_Y), \\
\forall f : X \to Y, & \quad Df : X \times X \to Y.
\end{align*}
\]
Cartesian differential categories [Blute, Cockett, Seely ’09]

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

1. $\mathcal{C}$ is Cartesian (has $\times$ and a terminal object $1$),
2. every object has a chosen commutative monoid structure, meaning there are $0_X : 1 \to X$ and $+_X : X \times X \to X$ 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, $\mathcal{C}$ is a Cartesian differential category means:

1. $\mathcal{C}$ is Cartesian (has $\times$ and a terminal object $1$),
2. every object has a chosen commutative monoid structure, meaning there are $0_X : 1 \to X$ and $+_X : X \times X \to X$ 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 \( 0_X : 1 \to X \) and \( +_X : X \times X \to X \) 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 \( f : X \to Y \) to \( Df : X \times X \to Y \) satisfying seven axioms...
A Cartesian differential operator sending \( f : X \to Y \) to \( Df : X \times X \to Y \) satisfies these axioms:

**CD1.** \( Ds = s \) for \( s \in \{ \text{id}_X, \sigma_{X,Y}, !_X, \Delta_X, 0_X, +X \} \).

**CD2.** \( Df 0 = 0 \)

**CD3.** \( Df +Df = Df +Df \)

**CD4.** \( D(fg) = DfDg \)

**CD5.** \( D(fg) = DgDf \)
Cartesian differential operator axioms, continued

CD6. 0

\[ D D f \quad = \quad D f \]

CD7. \( \otimes \)

\[ D D f \quad = \quad D D f \]
Cartesian differential operator axioms, continued

\[ CD6. \quad D D f \quad = \quad D f \]

\[ CD7. \quad D D f \quad = \quad D D f \]

**Example**

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

CD6. \[ DDf = Df \]

CD7. \[ DDf = DDf \]

Example

Objects of the category \( \text{Euc}_\infty \) are \( \mathbb{R}^n \) for \( n \in \mathbb{N} \), maps are smooth functions between them. \( \text{Euc}_\infty \) 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 \( \mathcal{C} \) is a Cartesian differential category. Can we give \( \text{St}(\mathcal{C}) \) a differential operator as well?
Differentiating RNN unrollings

\[ D(\psi) = D\psi \]

\[ D(\psi) = D\psi \]

\[ D(\psi) = D\psi \]
What could this be the unrolling of?

\[
D\left(\begin{array}{c}
i \\
\psi \\
\psi \\
\psi
\end{array}\right) =
\begin{array}{c}
D\psi \\
D\psi \\
D\psi
\end{array}
\]
What could this be the unrolling of?

\[ D(\psi \rightarrow \psi \rightarrow \psi ) = D\psi \rightarrow D\psi \rightarrow D\psi \]

Enough staring suggests the *recurrence rule*:

\[ D^* (\psi \rightarrow i) \symtriangleq (\psi \rightarrow i) \rightarrow D^* \psi \]
Main contributions

**Theorem: causal differentiation**

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

(The differential operator $D^\ast$ is basically $D$ augmented with the recurrence rule.)
Main contributions

Theorem: causal differentiation

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

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

Theorem: categorical BPTT

\[
DU_k(\Psi) = U_k(D^*\Psi) \circ z_k \quad \text{for all} \quad k \in \mathbb{N} \quad \text{and} \quad \Psi \in \text{St}(\mathcal{C})(A, B).
\]

\((z : (\prod X_i) \times (\prod Y_i) \rightarrow \prod (X_i \times Y_i) \text{ is a zipping isomorphism.})\)
Main contributions

**Theorem: causal differentiation**

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

(The differential operator $D^*$ is basically $D$ augmented with the recurrence rule.)

**Theorem: categorical BPTT**

$DU_k(\Psi) = U_k(D^*\Psi) \circ z_k$ for all $k \in \mathbb{N}$ and $\Psi \in \text{St}(\mathcal{C})(\mathbf{A}, \mathbf{B})$. 

($z : (\prod X_i) \times (\prod Y_i) \to \prod(X_i \times Y_i)$ is a zipping isomorphism.)

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

\[ D^* (\phi_i \psi_j) \]
Example: stateful chain rule

$\mathcal{D}^*(\phi_i \psi_j) = \mathcal{D}^\ast \phi \otimes \mathcal{D}^\ast \psi$
Example: stateful chain rule

\[ D^*(\begin{array}{cc} \phi & i \\ \psi & j \end{array}) = \]
Example: stateful chain rule

\[ \mathcal{D}^*(\phi_i \psi_j) = \]

\[ \mathcal{D}^*(\phi \psi) \]

\[ \mathcal{D}^* \phi \]

\[ \mathcal{D}^* \psi \]
Example: stateful chain rule

\[ D^*(\phi_i \psi_j) = \]

\[ \parallel \]

\[ D^*(\phi \psi) = \]

\[ D^*\phi \]

\[ D^*\psi \]
Example: stateful chain rule

$$\mathcal{D}^*(\phi_i \psi_j) =$$

$$\mathcal{D}^*(\phi_i \psi_j) =$$
Given a Cartesian differential category $\mathcal{C}$, we can construct $\text{St}(\mathcal{C})$, another CDC with stateful functions (delayed trace).
1. Given a Cartesian differential category $\mathcal{C}$, we can construct $\text{St}(\mathcal{C})$, another CDC with stateful functions (delayed trace).

2. Differentiation for stateful functions includes a recurrence rule:

$$D^* \triangleq \psi_0$$

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

Category theory and compositionality can be useful tools to organize and find structure in machine learning.
Given a Cartesian differential category $\mathbb{C}$, we can construct $\text{St}(\mathbb{C})$, another CDC with stateful functions (delayed trace).

Differentiation for stateful functions includes a recurrence rule:

$$D^* \triangleq \psi_i D^* \psi \psi_0$$

Stateful differentiation is connected to BPTT, and we can find many useful properties of BPTT with this connection.
Given a Cartesian differential category $\mathcal{C}$, we can construct $\text{St}(\mathcal{C})$, another CDC with stateful functions (delayed trace).

Differentiation for stateful functions includes a recurrence rule:

$$D^* \psi \triangleq D^* \psi_0$$

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

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 $\phi \in \{\text{LSTM, GRU, ...}\}$?
12. Implementation/plugin to neural net library?
References


Thanks!