Neural Datalog Through Time

Hongyuan Mei¹, Guanghui Qin¹, Minjie Xu², Jason Eisner¹
¹Johns Hopkins University
²Bloomberg
Model How a **Database** Changes Over Time
Model How a Database Changes Over Time

200,000 facts right now

relation(eve, adam)

at(eve, nyc)

at(adam, chicago)

opinion(eve, apples)

opinion(adam, apples)
Model How a **Database** Changes Over Time

200,000 facts right now

relation(eve, adam)  
at(eve, nyc)  
at(adam, chicago)  
claim(eve, apples)  
claim(adam, apples)
Model How a Database Changes Over Time

200,000 facts right now
50,000 possible events right now

relation(eve, adam)
travel(eve, chicago)
at(eve, nyc)
at(adam, chicago)
opinion(eve, apples)
opinion(adam, apples)
Model How a Database Changes Over Time

200,000 facts right now
50,000 possible events right now
Model How a **Database** Changes Over Time

200,000 facts right now
50,000 possible events right now

little language to specify a generative model of event sequences

- travel(eve, chicago)
- at(eve, nyc)
- relation(eve, adam)
- at(adam, chicago)
- opinion(eve, apples)
- opinion(adam, apples)
Model How a Database Changes Over Time

200,000 facts right now
50,000 possible events right now

little language to specify a generative model of event sequences
Model How a **Database** Changes Over Time

200,000 facts right now

50,000 possible events right now

**little language**

to specify a generative model
of event sequences
Model How a **Database** Changes Over Time

200,000 facts right now

50,000 possible events right now

little language to specify a generative model of event sequences

triggering rules

relation(eve, adam)

opinion(eve, apples)

opinion(adam, apples)

travel(eve, chicago)

at(eve, chicago)

at(adam, chicago)
Model How a Database Changes Over Time

200,000 facts right now
50,000 possible events right now

little language to specify a generative model of event sequences
Model How a **Database** Changes Over Time

200,000 facts right now
50,000 possible events right now

little language to specify a generative model of event sequences
Model How a Database Changes Over Time

200,000 facts right now

50,000 possible events right now

little language to specify a generative model of event sequences

opinion(eve, apples)

relation(eve, adam)

at(eve, chicago)

at(adam, chicago)

dinner(eve, adam)

opinion(adam, apples)
Model How a **Database** Changes Over Time

200,000 facts right now
50,000 possible events right now

deductive rules

relation(eve, adam)
opinion(eve, apples)
opinion(adam, apples)
at(eve, chicago)
at(adam, chicago)
dinner(eve, adam)

little language to specify a generative model of event sequences
Model How a **Database** Changes Over Time

200,000 facts right now
50,000 possible events right now

little language to specify a generative model of event sequences
Model How a **Database** Changes Over Time

200,000 facts right now
50,000 possible events right now

**deductive rules**

little language to specify a generative model of event sequences
Model How a **Database** Changes Over Time

200,000 facts right now
50,000 possible events right now

little language to specify a generative model of event sequences
Deductive Rules, Triggering Rules

\[
\begin{align*}
\text{relation}(X, Y) & \quad \text{:- opinion}(X, U), \text{opinion}(Y, U). \\
\text{travel}(X, P) & \quad \text{:- relation}(X, Y), \text{at}(Y, P). \\
!\text{at}(X, Q) & \quad \leftarrow \text{travel}(X, P), \text{at}(X, Q), P \neq Q. \\
\text{at}(X, P) & \quad \leftarrow \text{travel}(X, P). \\
\text{dinner}(X, Y) & \quad \text{:- relation}(X, Y), \text{at}(X, P), \text{at}(Y, P). \\
\text{relation}(X, Y) & \quad \leftarrow \text{dinner}(X, Y).
\end{align*}
\]
Deductive Rules, Triggering Rules

relation(X, Y) :- opinion(X, U), opinion(Y, U).

travel(X, P) :- relation(X, Y), at(Y, P).

!at(X, Q) ← travel(X, P), at(X, Q), P != Q.

at(X, P) ← travel(X, P).

dinner(X, Y) :- relation(X, Y), at(X, P), at(Y, P).

relation(X, Y) ← dinner(X, Y).
Deductive Rules, Triggering Rules

which facts are in the database

**logic!**

relation(X, Y)

::= opinion(X, U), opinion(Y, U).

travel(X, P)

::= relation(X, Y), at(Y, P).

!at(X, Q)

⇐ travel(X, P), at(X, Q), P != Q.

at(X, P)

⇐ travel(X, P).

dinner(X, Y)

::= relation(X, Y), at(X, P), at(Y, P).

relation(X, Y)

⇐ dinner(X, Y).
which facts are in the database
define a trainable neural architecture

**Deductive Rules, Triggering Rules**

**logic!**

<table>
<thead>
<tr>
<th>Relation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>relation(X, Y)</td>
<td>:- opinion(X, U), opinion(Y, U).</td>
</tr>
<tr>
<td>travel(X, P)</td>
<td>:- relation(X, Y), at(Y, P).</td>
</tr>
<tr>
<td>!at(X, Q)</td>
<td>&lt;- travel(X, P), at(X, Q), P != Q.</td>
</tr>
<tr>
<td>at(X, P)</td>
<td>&lt;- travel(X, P).</td>
</tr>
<tr>
<td>dinner(X, Y)</td>
<td>:- relation(X, Y), at(X, P), at(Y, P).</td>
</tr>
<tr>
<td>relation(X, Y)</td>
<td>&lt;- dinner(X, Y).</td>
</tr>
</tbody>
</table>
which facts are in the database define a trainable neural architecture that computes embeddings of the facts

**Deductive Rules, Triggering Rules**

<table>
<thead>
<tr>
<th>Relation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>relation(X, Y)</td>
<td>:- opinion(X, U), opinion(Y, U).</td>
</tr>
<tr>
<td>travel(X, P)</td>
<td>:- relation(X, Y), at(Y, P).</td>
</tr>
<tr>
<td>!at(X, Q)</td>
<td>&lt;- travel(X, P), at(X, Q), P != Q.</td>
</tr>
<tr>
<td>at(X, P)</td>
<td>&lt;- travel(X, P).</td>
</tr>
<tr>
<td>dinner(X, Y)</td>
<td>:- relation(X, Y), at(X, P), at(Y, P).</td>
</tr>
<tr>
<td>relation(X, Y)</td>
<td>&lt;- dinner(X, Y).</td>
</tr>
</tbody>
</table>
Deductive Rules, Triggering Rules

relation(X, Y) :- opinion(X, U), opinion(Y, U).

travel(X, P) :- relation(X, Y), at(Y, P).

!at(X, Q) \iff travel(X, P), at(X, Q), P \neq Q.

at(X, P) \iff travel(X, P).

dinner(X, Y) :- relation(X, Y), at(X, P), at(Y, P).

relation(X, Y) \iff dinner(X, Y).
which facts are in the database define a trainable neural architecture that computes embeddings of the facts

Deductive Rules, Triggering Rules

logic!

relation(X, Y)  :-  opinion(X, U), opinion(Y, U).
travel(X, P)  :-  relation(X, Y), at(Y, P).
!at(X, Q)  \leftarrow  travel(X, P), at(X, Q), P \neq Q.
at(X, P)  \leftarrow  travel(X, P).
dinner(X, Y)  :-  relation(X, Y), at(X, P), at(Y, P).
relation(X, Y)  \leftarrow  dinner(X, Y).
Deductive Rules, Triggering Rules

which facts are in the database define a trainable neural architecture that computes embeddings of the facts

relation(X, Y)
    :- opinion(X, U), opinion(Y, U).

travel(X, P)
    :- relation(X, Y), at(Y, P).

!at(X, Q)
    ` travel(X, P), at(X, Q), P != Q.

at(X, P)
    ` travel(X, P).

dinner(X, Y)
    :- relation(X, Y), at(X, P), at(Y, P).

relation(X, Y)
    ` dinner(X, Y).
Datalog ➔ Neural Datalog Through Time
Datalog $\rightarrow$ **Neural Datalog Through Time**

deductive rule

new fact :- old fact $^1$, old fact $^2$, ...

Datalog $\Rightarrow$ **Neural Datalog** Through Time

deductive rule

add to database

new fact :- old fact \(1\), old fact \(2\), ...
Datalog $\rightarrow$ **Neural** Datalog **Through Time**

**deductive rule**

add to database if

new fact $:\neg$ old fact $_1$, old fact $_2$, ...

If you're interested in a specific topic, feel free to ask! I'm here to help.
Datalog $\Rightarrow$ **Neural Datalog** Through Time

deductive rule

**add to database** if these are in database

new fact $\leftarrow$ old fact $^1$, old fact $^2$, ...

likes (X, U), likes (Y, U)
Datalog $\Rightarrow$ **Neural Datalog** Through Time

deductive rule

add to database if these are in database

new fact $:=$ old fact ₁, old fact ₂, ...

likes(X, U),
Datalog $\Rightarrow$ Neural Datalog Through Time

deductive rule

new fact :- old fact \(_1\), old fact \(_2\), ...

likes(X, U), likes(Y, U)

add to database if these are in database
Datalog $\Rightarrow$ **Neural Datalog** Through Time

deductive rule

add to database if these are in database

new fact :- old fact \(_1\), old fact \(_2\), ...

:- likes(X, U), likes(Y, U)
Datalog $\Rightarrow$ **Neural** Datalog **Through Time**

deductive rule

add to database if these are in database

new fact $\Leftarrow$ old fact $^1$, old fact $^2$, ...

compatible($X$, $Y$) $\Leftarrow$ likes($X$, $U$), likes($Y$, $U$)
Datalog $\Rightarrow$ **Neural Datalog Through Time**

**Deductive Rule**

Add to database if these are in database

\[
\text{new fact} ::= \text{old fact}_1, \text{old fact}_2, \ldots
\]

compatible\((X, Y) ::= \text{likes}(X, U), \text{likes}(Y, U)\)

\[
\text{compatible}(\text{eve, adam})
\]

\[
\text{likes}(\text{eve, apples}) \quad \text{likes}(\text{adam, apples})
\]
Datalog $\Rightarrow$ Neural Datalog Through Time

deductive rule

add to database if these are in database

new fact :- old fact $^1$, old fact $^2$, ...

compatible(X, Y) :- likes(X, U), likes(Y, U)

compatible(eve, adam)

likes(eve, apples)  
likes(adam, apples)
Datalog ➔ Neural Datalog Through Time

deductive rule

add to database if these are in database

new fact :- old fact ₁, old fact ₂, ...

compatible(X, Y) :- likes(X, U), likes(Y, U)

compatible(eve, adam)

likes(eve, apples) likes(adam, apples)
Datalog $\Rightarrow$ **Neural Datalog Through Time**

deductive rule

add to database if these are in database

new fact :- old fact \(_1\), old fact \(_2\), ...

compatible(X, Y) :- likes(X, U), likes(Y, U)

relation(eve, adam)

opinion(eve, apples)

opinion(adam, apples)
Datalog $\Rightarrow$ **Neural Datalog Through Time**

**deductive rule**

add to database if these are in database

new fact :- old fact $^1$, old fact $^2$, ...

**triggering rule**

new fact $\leftarrow$ event, old fact $^1$, old fact $^2$, ...
Datalog $\rightarrow$ **Neural Datalog Through Time**

**Deductive Rule**

\[
\text{add to database if these are in database}
\]

\[
\text{new fact} \leftarrow \text{old fact}_1, \text{old fact}_2, \ldots
\]

**Triggering Rule**

\[
\text{when new fact} \leftarrow \text{event, old fact}_1, \text{old fact}_2, \ldots
\]
Datalog $\Rightarrow$ Neural Datalog Through Time

deductive rule

add to database if these are in database

new fact :- old fact $^1$, old fact $^2$, ...

triggering rule

when this happens

new fact $\leftarrow$ event, old fact $^1$, old fact $^2$, ...
Datalog ➔ Neural Datalog Through Time

deductive rule
add to database if these are in database
new fact :¬ old fact₁, old fact₂, ...

triggering rule
when this happens while these are in database
new fact ← event, old fact₁, old fact₂, ...
Datalog $\Rightarrow$ Neural Datalog Through Time

deductive rule

add to database if these are in database

new fact $\leftarrow$ old fact \(_1\), old fact \(_2\), ... 

triggering rule

add to database when this happens while these are in database

new fact $\leftarrow$ event, old fact \(_1\), old fact \(_2\), ...
Datalog ➔ **Neural Datalog** Through Time

**deductive rule**

add to database if these are in database

new fact \( \leftarrow \) old fact \( 1 \), old fact \( 2 \), ...

**triggering rule**

add to database when this happens while these are in database

new fact \( \leftarrow \) event, old fact \( 1 \), old fact \( 2 \), ...

! old fact \( \leftarrow \) event, old fact \( 1 \), old fact \( 2 \), ...
### Datalog \( \rightarrow \) Neural Datalog Through Time

**deductive rule**

```
add to database if these are in database
new fact :- old fact \(_1\), old fact \(_2\), ...
```

**triggering rule**

```
add to database when this happens while these are in database
new fact \(\leftarrow\) event, old fact \(_1\), old fact \(_2\), ...
```

```
when ! old fact \(\leftarrow\) event, old fact \(_1\), old fact \(_2\), ...
```
**Datalog ➔ Neural Datalog Through Time**

**deductive rule**

add to database if these are in database

\[ \text{new fact} \leftarrow \text{old fact}_1, \text{old fact}_2, \ldots \]

**triggering rule**

add to database when this happens while these are in database

\[ \text{new fact} \leftarrow \text{event}, \text{old fact}_1, \text{old fact}_2, \ldots \]

when this happens

\[ \text{! old fact} \leftarrow \text{event}, \text{old fact}_1, \text{old fact}_2, \ldots \]
Datalog ➔ **Neural Datalog Through Time**

**deductive rule**

```
add to database if these are in database
new fact :- old fact₁, old fact₂, ...
```

**triggering rule**

```
add to database when this happens while these are in database
new fact ← event, old fact₁, old fact₂, ...
```

```
when this happens while these are in database
! old fact ← event, old fact₁, old fact₂, ...
```
Datalog ➔ Neural Datalog Through Time

deductive rule
add to database if these are in database
new fact :- old fact ₁, old fact ₂, ...

triggering rule
add to database when this happens while these are in database
new fact ← event, old fact ₁, old fact ₂, ...

delete when this happens while these are in database
! old fact ← event, old fact ₁, old fact ₂, ...
Computing the Embeddings
Computing the Embeddings

relation(eve, adam)
Computing the Embeddings

\[
\text{relation}(X, Y) :- \text{opinion}(X, U), \text{opinion}(Y, U)
\]

\[
\text{relation}(\text{eve}, \text{adam})
\]
Computing the **Embeddings**

relation(X, Y) :- opinion(X, U), opinion(Y, U)

relation(eve, adam)
Computing the **Embeddings**

\[ \text{relation}(X, Y) :- \text{opinion}(X, U), \text{opinion}(Y, U) \]

- \text{relation}(\text{eve}, \text{adam})
- \text{opinion}(\text{adam}, \text{apples})
- \text{opinion}(\text{eve}, \text{apples})
Computing the **Embeddings**

\[
\text{relation}(X, Y) ::= \text{opinion}(X, U), \text{opinion}(Y, U)
\]

- \text{opinion}(eve, apples)
- \text{opinion}(adam, apples)
- \text{relation}(eve, adam)
Computing the **Embeddings**

relation(X, Y) :- opinion(X, U), opinion(Y, U)

relation(eve, adam)

opinion(adam, apples)

opinion(eve, apples)
Computing the Embeddings

relation(X, Y) :- opinion(X, U), opinion(Y, U)

relation(eve, adam)

opinion(adam, apples)

opinion(eve, apples)
Computing the **Embeddings**

relation$(X, Y) :-$ opinion$(X, U),$ opinion$(Y, U)$

$$= \begin{pmatrix} 
\ast & \ast & \ast \\
\ast & \ast & \ast \\
\ast & \ast & \ast 
\end{pmatrix} \times \begin{pmatrix} 
\ast \\
\ast \\
\ast 
\end{pmatrix} + \begin{pmatrix} 
\ast & \ast & \ast \\
\ast & \ast & \ast \\
\ast & \ast & \ast 
\end{pmatrix} \times \begin{pmatrix} 
\ast \\
\ast \\
\ast 
\end{pmatrix}$$

relation$(eve, adam)$

opinion$(adam, apples)$

opinion$(eve, apples)$
Computing the **Embeddings**

\[
\text{relation}(X, Y) :- \text{opinion}(X, U), \text{opinion}(Y, U)
\]

\[
= x + x
\]

\[
\text{relation}(\text{eve, adam})
\]

\[
\text{opinion}(\text{adam, apples})
\]

\[
\text{opinion}(\text{eve, apples})
\]
Computing the Embeddings

relation(X, Y) :- opinion(X, U), opinion(Y, U)

= x + x

relation(eve, adam)

opinion(eve, apples) -> opinion(adam, apples)

opinion(eve, politics) -> opinion(adam, politics)
Computing the **Embeddings**

\[
\text{relation}(X, Y) :\text{ opinion}(X, U), \text{ opinion}(Y, U) = \text{ opinion}(X, U) \times \text{ opinion}(Y, U)
\]

relation(eve, adam)
Computing the **Embeddings**

relation\((X, Y)\) :- \(\text{opinion}(X, U)\), \(\text{opinion}(Y, U)\)

\[
\text{relation}(X, Y) = \text{opinion}(X, U) \times \text{opinion}(Y, U)
\]

\[
\text{relation}(\text{eve}, \text{adam}) = \text{opinion}(\text{eve}, \text{apples}) \times \text{opinion}(\text{adam}, \text{apples}) + \text{opinion}(\text{adam}, \text{politics}) \times \text{opinion}(\text{eve}, \text{politics})
\]
Computing the Embeddings

relation(X, Y) :- opinion(X, U), opinion(Y, U)

relation(eve, adam)

opinion(eve, apples)

opinion(adam, apples)

opinion(adam, politics)

opinion(eve, politics)

opinion(eve, politics)
Computing the Embeddings

relation(X, Y) :- opinion(X, U), opinion(Y, U)

\[ x + x \]

different inputs

same params
Computing the **Embeddings**

relation(X, Y) :- opinion(X, U), opinion(Y, U)

\[ \text{relation}(X, Y) = x + x \]

different inputs

same params

married(eve, adam)

opinion(eve, apples)

opinion(adam, apples)

opinion(eve, politics)

opinion(adam, politics)
Computing the **Embeddings**

relation(X, Y) :: opinion(X, U), opinion(Y, U)

\[ x + x \]

different inputs

same params

married(eve, adam)

non-linear pooling

opinion(eve, apples)

opinion(adam, apples)

opinion(eve, politics)

opinion(adam, politics)
Computing the **Embeddings**

\[
\text{relation}(X, Y) := \text{opinion}(X, U), \text{opinion}(Y, U) = x + x
\]

different inputs

\[
\text{married}(eve, adam)
\]

same params

\[
\text{LSTM cells: summarize past events that are relevant to this fact}
\]

\[
\sigma( + )
\]

non-linear pooling

\[
\text{opinion}(eve, apples)
\]

\[
\text{opinion}(adam, apples)
\]

\[
\text{opinion}(adam, politics)
\]

\[
\text{opinion}(eve, politics)
\]
Computing Embeddings & Probabilities
Computing **Embeddings** & **Probabilities**

- relation(eve, adam)
- relation(eve, cain)
- ...
Computing **Embeddings** & **Probabilities**

```
travel(X, P) :- relation(X, Y), at(Y, P).
```

---

```
relation(eve, adan)  relation(eve, cain)
```

...
Computing **Embeddings** & **Probabilities**

\[
\text{travel}(X, P) :- \text{relation}(X, Y), \text{at}(Y, P).
\]

- relation(eve, adam)
- relation(eve, cain)
- at(adam, chicago)
- at(cain, chicago)
Computing **Embeddings & Probabilities**

```
travel(X, P) :- relation(X, Y), at(Y, P).
```

![Diagram showing relationships between entities and relations]

- `travel(eve, chicago)`
- `relation(eve, adam)`
- `relation(eve, cain)`
- `at(adam, chicago)`
- `at(cain, chicago)`
Computing **Embeddings & Probabilities**

\[
\text{travel}(X, P) :\text{ relation}(X, Y), \text{ at}(Y, P).
\]

\[
\begin{align*}
\text{relation}(X, Y) &= \text{relation}(X, Y) \\
\text{at}(Y, P) &= \text{at}(Y, P)
\end{align*}
\]
Computing Embeddings & Probabilities

\[
\text{travel}(X, P) ::= \text{relation}(X, Y), \text{at}(Y, P).
\]

- travel(eve, chicago)

**Non-linear pooling**

relation(eve, adam)  
\text{at}(adam, chicago)

relation(eve, cain)  
\text{at}(cain, chicago)
Computing Embeddings & Probabilities

\[ \text{travel}(X, P) :\text{ relation}(X, Y), \text{ at}(Y, P). \]

\[ = \sigma(\sigma(\text{travel}(X, P)) + \text{relation}(X, Y)) \times \text{at}(Y, P). \]

\[ \sigma(\text{relation}(X, Y)) \]

\[ \sigma(\text{at}(Y, P)) \]

non-linear pooling
Computing **Embeddings** & **Probabilities**

\[
\text{travel}(X, P) : \text{relation}(X, Y), \text{at}(Y, P).
\]

\[
= \sigma(\text{relation}(X, Y)) + \text{at}(Y, P).
\]

**non-linear pooling**

\[
\text{relation}(X, Y) = \text{relation}(X, Y) \times \text{relation}(X, Y) + \text{relation}(X, Y) \times \text{relation}(X, Y)
\]

\[
\text{at}(X, Y) = \text{at}(X, Y) \times \text{at}(X, Y) + \text{at}(X, Y) \times \text{at}(X, Y)
\]

\[
\text{travel}(X, Y) = \text{relation}(X, Y) \times \text{at}(Y, P) + \text{relation}(X, Y) \times \text{at}(Y, P)
\]

\[
\text{relation}(X, Y) = \text{relation}(X, Y) \times \text{relation}(X, Y) + \text{relation}(X, Y) \times \text{relation}(X, Y)
\]

\[
\text{at}(X, Y) = \text{at}(X, Y) \times \text{at}(X, Y) + \text{at}(X, Y) \times \text{at}(X, Y)
\]

\[
\text{travel}(X, Y) = \text{relation}(X, Y) \times \text{at}(Y, P) + \text{relation}(X, Y) \times \text{at}(Y, P)
\]
Computing **Embeddings & Probabilities**

$$\text{travel}(X, P) := \text{relation}(X, Y), \text{at}(Y, P).$$

\[
\text{travel}(\text{eve, chicago}) = \sigma(\dots) 
\]

\[
\text{relation}(\text{eve, adam}) + \text{relation}(\text{eve, cain}) 
\]

\[
\text{at}(\text{adam, chicago}) \quad \text{at}(\text{cain, chicago}) 
\]

**non-linear pooling**
Computing **Embeddings & Probabilities**

\[
\text{travel}(X, P) \gets \text{relation}(X, Y), \text{at}(Y, P).
\]

\[
= \sigma(\text{relation}(\text{eve, adam}) + \text{relation}(\text{eve, cain}))
\]

`...`

\[
\sigma(a + b)
\]

**non-linear pooling**

\[
\text{at}(\text{adam, chicago})
\]

\[
\text{at}(\text{cain, chicago})
\]
Rules ➔ Deep Recurrent Neural Net
Rules $\rightarrow$ Deep Recurrent Neural Net

deductive :- rules
Rules $\Rightarrow$ Deep Recurrent Neural Net

LSTM cells

+ deductive :- rules
Rules $\rightarrow$ Deep Recurrent Neural Net

LSTM cells

$$\sigma(\quad + \quad)$$

deductive :- rules
Rules \rightarrow \textbf{Deep Recurrent Neural Net}

\[ \sigma(\text{LSTM cells}) + \text{rules} \rightarrow \text{triggering} \]

\text{deductive} : - \text{rules}
Rules $\Rightarrow$ Deep Recurrent Neural Net

$a$ little like stacked LSTM

LSTM cells

triggering $\leftrightarrow$ rules

deductive :- rules
Rules $\rightarrow$ Deep Recurrent Neural Net

- a little like stacked LSTM
- LSTM cells
- triggering $\leftarrow$ rules
- deductive :- rules
- deep at a single time step
Rules ➔ Deep Recurrent Neural Net

A little like stacked LSTM

LSTM cells

triggering ← rules

temporally recurrent across time steps

deductive :- rules

deep at a single time step
Life Story of a Fact
Life Story of a Fact
Life Story of a **Fact**

new fact ← event, other facts
Life Story of a Fact

new fact ← event, other facts

fact added by event

time=0
Life Story of a Fact

new fact $\leftarrow$ event, other facts

fact added by event

LSTM cells

time=0
Life Story of a **Fact**

- New fact ← event, other facts
- Fact added by event

LSTM cells at time=0
Life Story of a Fact

new fact $\leftrightarrow$ event, other facts

embedding changes over time
probability changes over time
$\approx$ neural Hawkes process (Mei & Eisner, 2017)
Life Story of a Fact

- New fact ← event, other facts
- Fact added by event
- Updated
- Embedding changes over time
- Probability changes over time
- LSTM cells
- Time = 0

≈ neural Hawkes process (Mei & Eisner, 2017)
Life Story of a **Fact**

new fact ← \( \text{event, other facts} \)

LSTM cells

- fact added by event
- updated

embedding changes over time

probability changes over time

\( \approx \) neural Hawkes process (Mei & Eisner, 2017)
Life Story of a Fact

new fact $\leftarrow$ event, other facts

"update vector"
$\approx$ neural Hawkes process (Mei & Eisner, 2017)
Life Story of a Fact

new fact ← event, other facts

≈ neural Hawkes process (Mei & Eisner, 2017)
Life Story of a **Fact**

- **New Fact** \( \leftrightarrow \) **Event**, Other Facts
- **Old Fact** ! \( \leftrightarrow \) **Event**, Other Facts

- **Embedding** changes over time
- **Probability** changes over time

\[ \approx \text{Neural Hawkes Process (Mei & Eisner, 2017)} \]
Life Story of a Fact

- Fact added by event
- updated
- deleted

new fact $\leftarrow$ event, other facts

! old fact $\leftarrow$ event, other facts

LSTM cells

embedding changes over time

probability changes over time

$\approx$ neural Hawkes process (Mei & Eisner, 2017)
Life Story of a Fact

- New fact $\leftrightarrow$ event, other facts
- Old fact $\leftrightarrow$ event, other facts

LSTM cells

- Fact added by event
- Updated
- Deleted
- Added again

Embedding changes over time

Probability changes over time

$\approx$ neural Hawkes process (Mei & Eisner, 2017)
Life Story of a Fact

LSTM cells

- time=0
- fact added by event
- updated
- deleted
- added again

- new fact \(\leftrightarrow\) event, other facts
- old fact \(\leftrightarrow\) event, other facts

embedding changes over time
probability changes over time

\(\approx\) neural Hawkes process (Mei & Eisner, 2017)
Experiment: **Users watch TV programs**

collaborative filtering problem with timing

who watches what and when?
Experiment: **Users watch TV programs**

collaborative filtering problem with timing who watches what and when?

1000 users
Experiment: **Users watch TV programs**

collaborative filtering problem with timing
who watches what and when?

1000 users  49 TV programs to be released
Experiment: **Users watch TV programs**

collaborative filtering problem with timing
who watches what and when?

1000 users 49 TV programs to be released

49000 possible watch events
Experiment: **Users watch TV programs**

collaborative filtering problem with timing
who watches what and when?

1000 users  49 TV programs to be released

49000 possible *watch* events

can not watch it until it is released
Experiment: Users watch TV programs

collaborative filtering problem with timing who watches what and when?
1000 users 49 TV programs to be released
49000 possible watch events

can not watch it until it is released
Experiment: **Users watch TV programs**

collaborative filtering problem with timing who watches what and when?

1000 users 49 TV programs to be released

49000 possible watch events

can not watch it until it is released
Experiment: Users watch TV programs

collaborative filtering problem with timing who watches what and when?

1000 users 49 TV programs to be released

49000 possible watch events

can not watch it until it is released
Experiment: **Users watch TV programs**

collaborative filtering problem with timing who watches what and when?

1000 users 49 TV programs to be released

49000 possible watch events

can not watch it until it is released
Experiment: **Users watch TV programs**

collaborative filtering problem with timing who watches what and when?

1000 users  49 TV programs to be released

49000 possible watch events

can not watch it until it is released
Experiment: Users watch TV programs

collaborative filtering problem with timing
who watches what and when?

1000 users 49 TV programs to be released

49000 possible watch events

can not watch it until it is released
**Experiment: Users watch TV programs**

1000 users | 49 TV programs to be released

49,000 possible watch events

Collaborative filtering problem with timing:
Who watches what and when?

User (627) can not watch it until it is released.
Experiment: **Users watch TV programs**

collaborative filtering problem with timing
who watches what and when?

1000 users  49 TV programs to be released

49000 possible watch events

can not watch it until it is released
Experiment: Robots Kick/Pass/Steal
Experiment: **Robots Kick/Pass/Steal**

22 robot soccer players

`player(Number, Team)`
Experiment: **Robots Kick/Pass/Steal**

22 robot soccer players

player(Number, Team)
Experiment: **Robots Kick/Pass/Steal**

22 robot soccer players

player(Number, Team)

- player(5, a)
- player(6, a)
- player(13, b)
- player(17, b)
- has_ball(17)

kick only if has ball
Experiment: **Robots Kick/Pass/Steal**

22 robot soccer players

player(Number, Team)

- player(5, a)
- player(6, a)
- player(13, b)
- player(17, b)
- has_ball(17)

Kick only if has ball
Pass
Experiment: **Robots Kick/Pass/Steal**

22 robot soccer players

`player(Number, Team)`

- kick only if has ball
- pass only to a teammate

- player(5, a)
- player(6, a)
- player(13, b)
- player(17, b)
- has_ball(17)
Experiment: Robots Kick/Pass/Steal

22 robot soccer players

player(Number, Team)

kick only if has ball
pass only to a teammate
steal

player(5, a)
player(6, a)
player(13, b)
player(17, b)
has_ball(17)
Experiment: **Robots Kick/Pass/Steal**

22 robot soccer players

player(Number, Team)

- **Kick** only if has ball
- **Pass** only to a teammate
- **Steal** only from opponent

player(5, a)
player(6, a)
player(13, b)
player(17, b)

has_ball(17)
Experiment: **Robots Kick/Pass/Steal**

22 robot soccer players

- player(Number, Team)
- kick only if has ball
- pass only to a teammate
- steal only from opponent

- player(5, a)
- player(6, a)
- player(13, b)
- player(17, b)
- has_ball(17)
Experiment: **Robots Kick/Pass/Steal**

22 robot soccer players

player(Number, Team)

kick only if has ball
pass only to a teammate
steal only from opponent

player(5, a)

pass(17, 13)

player(6, a)

steal(6, 17)

player(13, b)

has_ball(17)

player(17, b)
Experiment: **Robots Kick/Pass/Steal**

22 robot soccer players

player(Number, Team)

- kick only if has ball
- pass only to a teammate
- steal only from opponent

```
player(5, a)  
pass(17, 13)  
steal(6, 17)  
has_ball(17)  

player(6, a)  

player(17, a)  

player(13, b)  

player(17, b)
```
Experiment: **Robots Kick/Pass/Steal**

22 robot soccer players

player(Number, Team)

- kick only if has ball
- pass only to a teammate
- steal only from opponent
Experiment: **Robots Kick/Pass/Steal**

22 robot soccer players

player(Number, Team)

- **Kick** only if has ball
- **Pass** only to a teammate
- **Steal** only from opponent
Experiment: **Robots Kick/Pass/Steal**

22 robot soccer players

- **kick** only if has ball
- **pass** only to a teammate
- **steal** only from opponent

player(Number, Team)

- player(5, a)
- player(6, a)
- player(13, b)
- player(17, b)

has_ball(13)
Experiment: **Robots Kick/Pass/Steal**

- 22 robot soccer players
- **player(Number, Team)**

- **kick** only if has ball
- **pass** only to a teammate
- **steal** only from opponent
Results: NDTT > Competitors

3 error metrics (in 3 columns): smaller is better

users watch TV programs

robots kick/pass/steal soccer ball
Good Generalization with Less Data

log-likelihood

Neural Datalog Through Time Oracle

Neural Hawkes process

# of training sequences
Summary: Deep Recurrent Net
Summary: Deep Recurrent Net

e.g., RNN LSTM discrete-time
Summary: Deep Recurrent Net

e.g., RNN, discrete-time
LSTM
neural Hawkes process, continuous-time
Summary: Deep Recurrent Net

hidden system state
e.g., RNN LSTM
discrete-time continuous-time
neural Hawkes process
Summary:

Deep Recurrent Net

hidden system state

\( h \in \mathbb{R}^{300} \)

e.g., RNN LSTM
discrete-time continuous-time

neural Hawkes process
Summary:

Deep Recurrent Net

hidden system state

\[ h \in \mathbb{R}^{300} \]

e.g., RNN  
LSTM  
neural Hawkes process  

\[ 0.5 \quad 0.2 \quad 0.3 \]

discrete-time  
continuous-time
Summary:

Deep Recurrent Net

hidden system state

$$h \in \mathbb{R}^{300}$$

e.g., RNN LSTM neural Hawkes process

discrete-time continuous-time
Summary:

Deep Recurrent Net

\[ h \in \mathbb{R}^{300} \]

hidden system state

0.5  
0.2

0.3 event!

recurrent update
e.g., RNN LSTM
neural Hawkes process continuous-time
Summary: Logic ➔ Deep Recurrent Net

hidden system state
Summary: Logic $\rightarrow$ Deep Recurrent Net

distributed hidden system state
Summary: Logic $\rightarrow$ Deep Recurrent Net

distributed

hidden system state

0.5 0.3 0.2
Summary: Logic $\rightarrow$ Deep Recurrent Net

distributed

hidden system state

II

database of logical facts + embeddings

0.5

0.3

0.2
Summary: Logic $\rightarrow$ Deep Recurrent Net

distributed

hidden system state

Ⅱ
database of logical facts + embeddings

0.5 0.3 0.2

event!
Summary: Logic $\rightarrow$ Deep Recurrent Net

distributed

hidden system state

$\parallel$

database of logical facts + embeddings

event!

recurrent update
Summary: Logic $\Rightarrow$ Deep Recurrent Net

distributed

hidden system state

II

database of logical facts + embeddings

0.5

0.3

depth

event!

recurrent

update
Summary: Logic $\Rightarrow$ **Deep Recurrent Net**

- **distributed**
- hidden system state
- database of logical facts + embeddings

**event!**

**update**

**update derived from small rule set + learned low-dim matrices**

new fact :- old fact \(1, \ldots\)

new fact $\Leftarrow$ event, \(\ldots\)

! old fact $\Leftarrow$ event, \(\ldots\)
Summary: Logic $\rightarrow$ Deep Recurrent Net

distributed
hidden system state

I I
database of logical facts + embeddings

0.5
0.3

new fact :- old fact $\bar{1}$, ...
new fact $\leftarrow$ event, ...
! old fact $\leftarrow$ event, ...

try our code in your domain!
Neural Datalog Through Time

Thank You

Bloomberg PhD Fellowship
Songyun Duan
Karan Uppal

ICLR reviewers
Songyun Duan
Hongteng Xu

ICML reviewers
Yujie Zha
Rakshit Trivedi

NSF
MARCC
NVIDIA
Argo Lab

JHU-CLSP