Neural Datalog Through Time: Informed Temporal Modeling via Logical Specification

Hongyuan Mei [1] **Guanghui Qin** [1] **Minjie Xu** [2] **Jason Eisner** [1]

Abstract

Learning how to predict future events from patterns of past events is difficult when the set of possible event types is large. Training an unrestricted neural model might overfit to spurious patterns. To exploit domain-specific knowledge of how past events might affect an event's present probability, we propose using a *temporal deductive database* to track structured facts over time. Rules serve to prove facts from other facts and from past events. Each fact has a time-varying state—a vector computed by a neural net whose topology is determined by the fact's *provenance*, including its experience of past events. The possible event types at any time are given by special facts, whose *probabilities* are neurally modeled alongside their states. In both synthetic and real-world domains, we show that neural probabilistic models derived from concise Datalog programs improve prediction by encoding appropriate domain knowledge in their architecture.

1. Introduction

Temporal sequences are abundant in applied machine learning. A common task is to predict the future from the past or to impute other missing events. Often this is done by fitting a generative probability model. For evenly spaced sequences, historically popular generative models have included hidden Markov models and discrete-time linear dynamical systems, with more recent interest in recurrent neural network models such as LSTMs. For irregularly spaced sequences, a good starting point is the Hawkes process (a self-exciting temporal point process) and its many variants, including neuralized versions based on LSTMs.

Under any of these models, each event e_i updates the state of the system from $\mathbf{s}_i$ to $\mathbf{s}_{i+1}$, which then determines the distribution from which the next event e_{i+1} is drawn. Alas, when the relationship between events and the system state is unrestricted—when anything can potentially affect anything—fitting an accurate model is very difficult, particularly in a real-world domain that allows millions of event types including many rare types. Thus, one would like to introduce domain-specific structure into the model.

For example, one might declare that the probability that Alice travels to Chicago is determined entirely by Alice's state, the states of Alice's coworkers such as Bob, and the state of affairs in Chicago. Given that modeling assumption, parameter estimation can no longer incorrectly overfit this probability using spurious features based on unrelated temporal patterns of (say) wheat sales and soccer goals.

To improve extrapolation, one can reuse this "Alice travels to Chicago" model for any person `A` traveling to any place `C`. Our main contribution is a modeling language that can concisely model all these `travel(A,C)` probabilities using a few rules over variables `A`, `B`, `C`. Here `B` ranges over `A`'s coworkers, where the `coworker` relation is also governed by rules and can itself be affected by stochastic events.

In our paradigm, a domain expert simply writes down the rules of a **temporal deductive database**, which tracks the possible event types and other *boolean* facts over time. This logic program is then used to automatically construct a deep recurrent neural architecture, whose distributed state consists of vector-space *embeddings* of all present facts. Its output specifies the distribution of the next event.

What sort of rules? An **event** has a *structured* description with zero or more participating **entities**. When an event happens, pattern-matching against its description triggers **update rules**, which modify the database facts to reflect the new properties and relationships of these entities. Updates may have a cascading effect if the database contains **deductive rules** that derive further facts from existing ones at any time. (For example, `coworker(A,B)` is jointly implied by `boss(U,A)` and `boss(U,B)`). In particular, deductive rules can state that entities combine into a possible event type whenever they have the appropriate properties and relationships. (For example, `travel(A,C)` is possible if `C` is a place and `A` is a person who is not already at `C`.)

[1]Computer Science Dept., Johns Hopkins Univ. [2]Bloomberg LP. Correspondence to: Hongyuan Mei <hmei@cs.jhu.edu>.

Proceedings of the 37^{th} International Conference on Machine Learning, Online, PMLR 119, 2020.

Since the database defines possible events and is updated by the event that happens, it already resembles the system state $\mathbf{s}_i$ of a temporal model. We enrich this logical state by associating an embedding with each fact currently in the database. This time-varying vector represents the state of *that fact*; recall that the set of facts may also change over time. When a fact is added by events or derived from other facts, its embedding is derived from their embeddings in a standard way, using parameters associated with the rules that established the fact. In this way, the model's rules together with the past events and the initial facts define the topology of a deep recurrent neural architecture, which can be trained via back-propagation through time (Williams & Zipser, 1989). For the facts that state that specific event types are possible, the architecture computes not only embeddings but also the probabilities of these event types.

The number of parameters of such a model grows only with the number of rules, not with the much larger number of event types or other facts. This is analogous to how a probabilistic relational model (Getoor & Taskar, 2007; Richardson & Domingos, 2006) derives a graphical model structure from a database, building random variables from database entities and repeating subgraphs with shared parameters.

Unlike graphical models, ours is a neural-symbolic hybrid. The system state $\mathbf{s}_i$ includes both rule-governed discrete elements (the set of facts) and learned continuous elements (the embeddings of those facts). It can learn a neural probabilistic model of people's movements while relying on a discrete symbolic deductive database to cheaply and accurately record who is where. A purely neural model such as our neural Hawkes process (Mei & Eisner, 2017) would have to *learn* how to encode every location fact in some very high-dimensional state vector, and retain and update it, with no generalization across people and places.

In our experiments, we show how to write down some domain-specific models for irregularly spaced event sequences in continuous time, and demonstrate that their structure improves their ability to predict held-out data.

2. Our Modeling Language

We gradually introduce our specification language by developing a fragment of a human activity model. Similar examples could be developed in many other domains—epidemiology, medicine, education, organizational behavior, consumer behavior, economic supply chains, etc. Such specifications can be trained and evaluated using our implementation, which can be found at `https://github.com/HMEIatJHU/neural-datalog-through-time`.

For pedagogical reasons, §2 will focus on our high-level scheme (see also the animated drawings in our ICML 2020 talk video). We defer the actual neural formulas until §3.

2.1. Datalog

We adapt our notation from Datalog (Ceri et al., 1989), where one can write **deductive rules** of the form

$$\mathit{head}\ \texttt{:-}\ \mathit{condit}_1\texttt{,}\ \ldots\texttt{,}\ \mathit{condit}_N\texttt{.} \tag{1}$$

Such a rule states that the head is true provided that the conditions are all true.[1] In a simple case, the head and conditions are **atoms**, i.e., structured terms that represent boolean propositions. For example,

```
compatible(eve,adam) :-
    likes(eve,apples), likes(adam,apples).
```

If $N = 0$, the rule simply states that the head is true. This case is useful to assert basic facts:

```
likes(eve,apples).
```

Notice that in this case, the :- symbol is omitted.

A rule that contains **variables** (capitalized identifiers) represents the infinite collection of **ground** rules obtained by instantiating (grounding) those variables. For example,

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

says that *any* two entities `X` and `Y` are compatible provided that there exists *any* `U` that they both like.

A Datalog **program** is an unordered set of rules. The atoms that can be proved from these rules are called **facts**. Given a program, one would use $[\![h]\!] \in \{\mathsf{true}, \mathsf{null}\}$ to denote the semantic value of atom h, where $[\![h]\!] = \mathsf{true}$ iff h is a fact.

2.2. Neural Datalog

In our formalism, a fact has an **embedding** in a vector space, so the semantic value of atom `likes(eve,apples)` describes more than just *whether* `eve` likes `apples`. To indicate this, let us rename and colorize the functors in rule 3:

```
rel(X,Y) :- opinion(X,U), opinion(Y,U).
```

Now $[\![$`opinion(eve,apples)`$]\!]$ is a vector describing `eve`'s complex opinion about apples (or null if she has no opinion). $[\![$`rel(eve,adam)`$]\!]$ is a vector describing `eve` and `adam`'s relationship (or null if they have none).

With this extension, $[\![h]\!] \in \mathbb{R}^{D_h} \cup \{\mathsf{null}\}$, where the embedding dimension D_h depends on the atom h. The declaration

```
:- embed(opinion,8).
```

says that if h has the form `opinion(...)` then $D_h = 8$.[2]

When an atom is proved via a rule, its embedding is affected by the conditions of that rule, in a way that depends on trainable parameters associated with that rule. For example, according to rule 4, $[\![$`rel(eve,adam)`$]\!]$ is a parametric function of the opinion vectors that `eve` and `adam` have about various topics `U`. The influences from all their shared topics are pooled together as detailed in §3.1 below.

[1] Appendix A.2 discusses an extension to negated conditions.

[2] In the absence of such a declaration, $D_h = 0$. Then $[\![h]\!]$ has only two possible values, just as in Datalog; we do not color h.

A model might say that *each* person has an opinion about *each* food, which is a function of the embeddings of the person and the food, using parameters associated with rule 6:

```
6 opinion(X,U) :- person(X), food(U).
```

If the foods are simply declared as basic facts, as follows, then each food's embedding is independently specified by the parameters associated with the rule that declares it:

```
7 food(apples).
8 food(manna).
      ⋮
```

Given all the rules above, whenever `person(X)` and `person(Y)` are facts, it follows that `rel(X,Y)` is a fact, and $[\![\texttt{rel(X,Y)}]\!]$ is defined by a multi-layer feed-forward neural network whose topology is given by the proof DAG for `rel(X,Y)`. The network details will be given in §3.1.

Recursive Datalog rules can lead to arbitrarily deep networks that recursively build up a compositional embedding, just as in sequence encoders (Elman, 1990), tree encoders (Socher et al., 2012; Tai et al., 2015), and DAG encoders (Goller & Kuchler, 1996; Le & Zuidema, 2015)—all of which could be implemented in our formalism. E.g.:

```
9  cursed(cain).
10 cursed(Y) :- cursed(X), parent(X,Y).
```

In Datalog, this system simply states that all descendants of `cain` are cursed. In neural Datalog, however, a child has a *specific* curse: a vector $[\![\texttt{cursed(Y)}]\!]$ that is computed from the parent's curse $[\![\texttt{cursed(X)}]\!]$ in a way that also depends on their relationship, as encoded by the vector $[\![\texttt{parent(X,Y)}]\!]$. Rule 10's parameters model how the curse evolves (and hopefully attenuates) as each generation is re-cursed. Notice that $[\![\texttt{cursed(Y)}]\!]$ is essentially computed by a recurrent neural network that encodes the sequence of `parent` edges that connect `cain` to `Y`.[3]

We currently consider it to be a model specification error if any atom h participates in its own proof, leading to a circular definition of $[\![h]\!]$. This would happen in rules 9–10 only if `parent` were bizarrely defined to make some cursed person their own ancestor. Appendix A.1 discusses extensions that would define $[\![h]\!]$ even in these cyclic cases.

2.3. Datalog Through Time

For temporal modeling, we use atoms such as `help(X,Y)` as the structured names for events. We underline their functors. As usual, we colorize them if they have vector-space embeddings (see footnote 2), but as orange rather than blue.

We extend Datalog with **update rules** so that whenever a `help(X,Y)` event occurs under appropriate conditions, it can add to the database by proving new atoms:

```
11 grateful(Y,X) <- help(X,Y), person(Y).
```

An event can also cancel out such additions, which may make atoms false again.[4] The `!` symbol means "not":

```
12 !grateful(Y,X) <- harm(X,Y).
```

The general form of these **update rules** is

$$\mathit{head} \leftarrow \mathit{event},\ \mathit{condit}_1,\ \ldots,\ \mathit{condit}_N. \quad (2a)$$

$$!\mathit{head} \leftarrow \mathit{event},\ \mathit{condit}_1,\ \ldots,\ \mathit{condit}_N. \quad (2b)$$

which state that event makes head true or false, respectively, provided that the conditions are all true. An event occurring at time s affects the set of facts at times $t > s$, both directly through $\leftarrow$ rules, and also indirectly, since the facts added or removed by $\leftarrow$ rules may affect the set of additional facts that can be derived by :- rules at time t. Our approach can be used for either discrete time ($s, t \in \mathbb{N}$) or continuous time ($s, t \in \mathbb{R}_{\geq 0}$), where the latter supports irregularly spaced events (e.g., Mei & Eisner, 2017).

2.4. Neural Datalog Through Time

In §2.2, we derived each fact's embedding from its proof DAG, representing its set of Datalog proofs. For Datalog through time, we must also consider how to embed facts that were proved by an earlier update. Furthermore, once an atom is proved, an update rule can prove it again. This will update its embedding, in keeping with our principle that a fact's embedding is influenced by *all* of its proofs.

As an example, when `X` helps `Y` and `grateful(Y,X)` first becomes true via rule 11, the new embedding $[\![\texttt{grateful(Y,X)}]\!]$ is computed—using parameters associated with rule 11—from the embeddings of `help(X,Y)` and `person(Y)`. Those embeddings model the nature of the help and the state of person `Y`. (This was the main reason for rule 11 to include `person(Y)` as a condition.) Each time `X` helps `Y` again, $[\![\texttt{grateful(Y,X)}]\!]$ is further updated by rule 11, so this gratitude vector records the *history* of help. The updates are LSTM-like (see §3.3 for details).

In general, an atom's semantics can now vary over time and so should be denoted as $[\![h]\!](t)$: the **state** of atom h at time t, which is part of the overall database state. A :- rule as in equation (1) says that $[\![\mathit{head}]\!](t)$ depends parametrically on $\{[\![\mathit{condit}_i]\!](t) : 1 \leq i \leq N\}$. A $\leftarrow$ rule as in equation (2a) says that if event occurred at time $s < t$ and no events updating head occurred on the time interval (s, t), then $[\![\mathit{head}]\!](t)$ depends parametrically on its previous value[5] $[\![\mathit{head}]\!](s)$ along with $[\![\mathit{event}]\!](s)$, $\{[\![\mathit{condit}_i]\!](s) : 1 \leq i \leq N\}$, and the elapsed time $t - s$. We will detail the parametric formulas in §3.3.

[3] Assuming that this path is unique. More generally, `Y` might descend from `cain` by multiple paths. The computation actually encodes the DAG of *all* paths, by pooling over all of `Y`'s cursed parents at each step, just as rule 4 pooled over multiple topics.

[4] The atom will remain true if it remains provable by a :- rule, or is proved by another $\leftarrow$ rule at the same time.

[5] More precisely, it depends on the LSTM cells that contributed to that previous value, as we will see in §3.3.

Thus, $[\![\texttt{head}]\!](t)$ depends via :- rules on *head*'s *provenance* in the database at time t, and depends via <- rules on its *experience* of events at strictly earlier times.[6] This yields a neural architecture similar to a stacked LSTM: the :- rules make the neural network deep at a single time step, while the <- rules make it temporally recurrent across time steps. The network's irregular topology is defined by the :- and <- rules plus the events that have occurred.

2.5. Probabilistic Modeling of Event Sequences

Because events can **occur**, atoms that represent event types are special. They can be declared as follows:

```
13 :- event(help, 8).
```

Because the declaration is `event` rather than `embed`, at times when `help(X,Y)` is a fact, it will have a positive probability along with its embedding $[\![\texttt{help(X,Y)}]\!] \in \mathbb{R}^8$. This is what the underlined functor really indicates.

At times s when `help(X,Y)` is not a fact, the semantic value $[\![\texttt{help(X,Y)}]\!](s)$ will be null, and it will have neither an embedding nor a probability. At these times, it is simply not a possible event; its probability is effectively 0.

Thus, the model must include rules that establish the set of possible events as facts. For example, the rule

```
14 help(X,Y) :- rel(X,Y).
```

says if `X` and `Y` have a relationship, then `help(X,Y)` is true, meaning that events of the type `help(X,Y)` have positive probability (i.e., `X` can help `Y`). The embedding and probability are computed deterministically from $[\![\texttt{rel(X,Y)}]\!]$ using parameters associated with rule 14, as detailed in §3.2.

Now a neural-Datalog-through-time program specifies a probabilistic model over event sequences. Each stochastic event can update some database facts or their embeddings, as well as the probability distribution over possible next events. As §1 outlined, each *stochastic draw* from the next-event distribution results in a *deterministic update* to that distribution—just as in a recurrent neural network language model (Mikolov et al., 2010; Sundermeyer et al., 2012).

Our approach also allows the possibility of **exogenous** events that are not generated by the model, but are given externally. Our probabilistic model is then *conditioned* on these exogenous events. The model itself might have probability 0 of generating these event types at those times. Indeed, if an event type is to occur *only* exogenously, then the model should not predict any probability for it, so it should not be declared using `event`. We use a dashed underline for undeclared events since they have no probability.

For example, we might wish to use rules of the form *head* <- `earthquake(C),...` to model how an earthquake in city `C` tends to affect subsequent events, even if we do not care to model the *probabilities* of earthquakes. The *embeddings* of possible earthquake events can still be determined by parametric rules, e.g., `earthquake(C) :- city(C)`, if we request them by declaring `embed(earthquake,5)`.

[6] See §3.3 for the precise interaction of :- and <- rules.

2.6. Continuing the Example

In our example, the following rules are also plausible. They say that when `X` helps `Y`, this event updates the states of the helper `X` and the helpee `Y` and also the state of their relationship:

```
15 person(X) <- help(X,Y).
16 person(Y) <- help(X,Y)
17 rel(X,Y)  <- help(X,Y).
```

To enrich the model further, we could add (e.g.) `rel(X,Y)` as a condition to these rules. Then the update when `X` helps `Y` depends quantitatively on the state of their relationship.

There may be many other kinds of events observed in a human activity dataset, such as `sleep(X)`, `eat(X)`, `email(X,Y)`, `invite(X,Y)`, `hire(X,Y)`, etc. These can be treated similarly to `help(X,Y)`.

Our modeling architecture is intended to limit dependencies to those that are explicitly specified, just as in graphical models. However, the resulting independence assumptions may be too strong. To allow unanticipated influences back into the model, it can be useful to include a low-dimensional global state, which is updated by all events:

```
18 world <- help(X,Y).
```

⋮

`world` records a "public history" in its state, and it can be a condition for any rule. E.g., we can replace rule 14 with

```
19 help(X,Y) :- rel(X,Y), world.
```

so that `eve`'s probability of helping `adam` might be affected by the history of other individuals' interactions.

Eventually `eve` and `adam` may die, which means that they are no longer available to help or be helped:

```
20 die(X) :- person(X).
```

If we want `person(eve)` to then become false, the model cannot place that atom in the database with a :- rule like

```
21 person(eve).
```

which would ensure that `person(eve)` can *always* be proved. Instead, we use a <- rule that initially adds `person(eve)` to the database via a special event, `init`, that always occurs exogenously at time $t = 0$:

```
22 person(eve) <- init.
```

With this treatment, the following rule can remove `person(eve)` again when she dies:

```
23 !person(X) <- die(X).
```

The reader may enjoy extending this model to handle possessions, movement, tribal membership/organization, etc.

2.7. Finiteness

Under our formalism, any given model allows only a finite set of possible events. This is because a Datalog program's facts are constructed by using functors mentioned in the program, with arguments mentioned in the program,[7] and nesting is disallowed. Thus, the set of facts is finite (though perhaps much larger than the length of the program).

It is this property that will ensure in §3.2 that our probability model—which sums over all possible events—is well-defined. Yet this is also a limitation. In some domains, a model should not really place any *a priori* bound on the number of event types, since an infinite sequence may contain infinitely many distinct types—the number of types represented in the length-n prefix grows unboundedly with n. Even our running example should really support the addition of new entities: the event `procreate(eve,adam)` should result in a fact such as `person(cain)`, where `cain` is a newly allocated entity. Similarly, new species are allocated in the course of drawing a sequence from Fisher's (1943) species-sampling model or from a Chinese restaurant process; new words are allocated as a document is drawn from an infinite-vocabulary language model; and new real numbers are constantly encountered in a sequence of sensor readings. In these domains, no model can *prespecify* all the entities that can appear in a dataset. Appendix A.4 discusses potential extensions to handle these cases.

3. Formulas Associated With Rules

3.1. Neural Datalog

Recall from §2.1 that if h is a fact, it is provable by at least one `:-` rule in at least one way. For neural Datalog (§2.2), we then choose to define the embedding $[\![h]\!] \neq \mathsf{null}$ as

$$[\![h]\!] \stackrel{\text{def}}{=} \tanh\Big(\sum_r [h]_r^{:-}\Big) \in (-1,1)^{D_h} \tag{3}$$

where $[h]_r^{:-}$ represents the **contribution** of the r^{th} rule of the Datalog program. For example, $[\![$`opinion(eve,apples)`$]\!]$ receives non-zero contributions from *both* rule 2 and rule 6.[8] For a given `Y`, $[\![$`cursed(Y)`$]\!]$ may receive a non-zero contribution from rule 9, rule 10, or neither, according to whether `Y` is `cain` himself, a descendant of `cain`, or neither.

The contribution $[h]_r^{:-}$ has been pooled over all the ways (if any) that the r^{th} rule proves h. For example, for any entity `Y`, $[$`cursed(Y)`$]_{10}^{:-}$ needs to compute the *aggregate* effect of the curses that `Y` inherits through *all* of `Y`'s cursed parents `X` in rule 10. Similarly, $[$`rel(X,Y)`$]_4^{:-}$ computes the aggregate effect on the relationship from *all* of `X` and `Y`'s shared interests `U` in rule 4. Recall from §2.1 that a rule with variables represents a collection of ground rules obtained by instantiating those variables. We define its contribution by

$$[h]_r^{:-} \stackrel{\text{def}}{=} \bigoplus_{g_1,\ldots,g_N}^{\beta_r} \mathbf{W}_r \underbrace{[1; [\![g_1]\!]; \ldots; [\![g_N]\!]]}_{\text{concatenation of column vectors}} \in \mathbb{R}^{D_h} \tag{4}$$

where for the summation, we allow h `:-` g_1`,` `...,` g_N to range over all instantiations of the r^{th} rule such that the head equals h and $g_1, \ldots, g_N$ are all facts. There are only finitely many such instantiations (see §2.7). $\mathbf{W}_r$ is a conformable parameter matrix associated with the r^{th} rule. (Appendix B offers extensions that allow more control over how parameters are shared among and within rules.)

The pooling operator $\bigoplus^\beta$ that we used above is defined to aggregate a set of vectors $\{\mathbf{x}_1, \ldots, \mathbf{x}_M\}$:

$$\bigoplus_m^\beta \mathbf{x}_m \stackrel{\text{def}}{=} v^{-1}\Big(\sum_m v(\mathbf{x}_m)\Big) \tag{5}$$

Remarks: For any definition of function v with inverse v^{-1}, $\bigoplus^\beta$ has a unique identity element, $v^{-1}(\mathbf{0})$, which is also the result of pooling no vectors ($M=0$). Pooling a single vector ($M=1$) returns that vector—so when rule r proves h in only one way, the contribution of the $[\![g_i]\!]$ to $[\![h]\!]$ does not have to involve an "extra" nonlinear pooling step in equation (4), but only the nonlinear $\tanh$ in equation (3).

Given $\beta \neq 0$, we take v to be the differentiable function

$$v(\mathbf{x}) \stackrel{\text{def}}{=} \operatorname{sign}(\mathbf{x})\,|\mathbf{x}|^\beta \tag{6a}$$

$$v^{-1}(\mathbf{y}) = \operatorname{sign}(\mathbf{y})\,|\mathbf{y}|^{1/\beta} \tag{6b}$$

where all operations are applied elementwise. Now the result of aggregating no vectors is $\mathbf{0}$, so rules that achieve no proofs of h contribute nothing to equation (3). If $\beta = 1$, then v = identity and $\bigoplus^\beta$ is just summation. As $\beta \to \infty$, $\bigoplus^\beta$ emphasizes more extreme values, approaching a signed variant of max-pooling that chooses (elementwise) the argument with the largest absolute value. As a generalization, one could replace the scalar β with a vector $\boldsymbol{\beta}$, so that different dimensions are pooled differently. Pooling is scale-invariant: $\bigoplus_m^\beta \alpha\mathbf{x}_m = \alpha \bigoplus_m^\beta \mathbf{x}_m$ for $\alpha \in \mathbb{R}$.

For each rule r, we learn a scalar β_r,[9] and use $\bigoplus^{\beta_r}$ in (4).

3.2. Probabilities and Intensities

When a fact h has been declared by `event` to represent an event type, we need it to have not only an embedding but

[7] A rule such as `likes(adam,Y) :- likes(adam,eve)` might be able to prove that `adam` likes everyone, including infinitely many unmentioned entities. To preserve finiteness, such rules are illegal in Datalog. A Datalog rule must be **range-restricted**: any variable in the head must also appear in the body.

[8] Recall that we renamed `likes` in rule 2 to `opinion`.

[9] It can be parameterized as $\beta = \exp b > 0$ (ensuring that aggregating positive numbers exceeds their max), or as $\beta = 1 + b^2 \geq 1$ (ensuring that the aggregate of positive numbers also does not exceed their sum). Our present experiments do the latter.

also a positive probability. We extend our setup by appending an extra row to the matrix $\mathbf{W}_r$ in (4), leading to an extra element in the column vectors $[h]^{:-}_r$. We then pass only the first D_h elements of $\sum_r [h]^{:-}_r$ through $\tanh$, obtaining the same $[\![h]\!]$ as equation (3) gave before. We pass the one remaining element through an $\exp$ function to obtain $\lambda_h > 0$.

Recall that for neural Datalog through time (§2.4), all these quantities, including λ_h, vary with the time t. To model a discrete-time event sequence, define the **probability** of an event of type h at time step t to be proportional to $\lambda_e(t)$, normalizing over all event types that are possible then. This imitates the softmax distributions in other neural sequence models (Mikolov et al., 2010; Sundermeyer et al., 2012).

When time is continuous, as in our experiments (§6), we need instantaneous probabilities. We take $\lambda_h(t)$ to be the (Poisson) **intensity** of h at time t: that is, it models the limit as $dt \to 0^+$ of the expected *rate* of h on the interval $[t, t+dt)$ (i.e., the expected number of occurrences of h divided by dt). This follows the setup of the neural Hawkes process (Mei & Eisner, 2017). Also following that paper, we replace $\exp(x) > 0$ in the above definition of λ_h with the function $\text{softplus}_\tau(x) = \tau \log(1 + \exp(x/\tau)) > 0$. We learn a separate temporal scale parameter τ for each functor and use the one associated with the functor of h.

In both discrete and continuous time, the exact model likelihood (§4) will involve a summation (at each time t) over the finite set of event types (§2.7) that are possible at time t.

Appendix A.6 offers an extension to simultaneous events.

3.3. Updates Through Time

We now add an LSTM-like component so that each atom will track the sequence of events that it has "seen"—that is, the sequence of events that updated it via <- rules (§2.3). Recall that an LSTM is constructed from **memory cells** that can be increased or decreased as successive inputs arrive.

Every atom h has a **cell block** $\boxed{h} \in \mathbb{R}^{D_h} \cup \{\text{null}\}$. When $\boxed{h} \neq \text{null}$, we augment h's embedding formula (3) to[10]

$$[\![h]\!] \stackrel{\text{def}}{=} \tanh\left(\boxed{h} + \sum_r [h]^{:-}_r\right) \in (-1, 1)^{D_h} \quad (7)$$

Properly speaking, $[\![h]\!]$, $\boxed{h}$, and $[h]^{:-}_r$ are all functions of t.

At times when $\boxed{h} = \text{null}$, we like to say that h is **docked**. Every atom h is docked initially (at $t = 0$), but may be **launched** through an update of type (2a), which ensures that $\boxed{h} \neq \text{null}$ and thus $[\![h]\!] \neq \text{null}$ by (7). h is subsequently **adrift** (and remains a fact) until it is docked again through an update of type (2b), which sets $\boxed{h} = \text{null}$.

[10] Recall from §3.2 that if h is an event, we extend $[\![h]\!]$ with an extra dimension to carry the probability. For equation (7) to work, we must likewise extend $\boxed{h}$ with an extra cell (when $\boxed{h} \neq \text{null}$).

How is $\boxed{h}$ updated by an event (or events[11]) occurring at time s? Suppose the r^{th} rule is an update rule of type (2a). Consider its instantiations h <- e, $g_1, \ldots, g_N$ (if any) with head h, such that e occurred at time s and $g_1, \ldots, g_N$ are all facts at time s. For the m^{th} instantiation, define

$$[h]^{\leftarrow}_{rm} \stackrel{\text{def}}{=} \mathbf{W}_r \underbrace{[1; [\![e]\!]; [\![g_1]\!]; \ldots; [\![g_N]\!]]}_{\text{concatenation of column vectors}} \quad (8)$$

where all embeddings are evaluated at time s, and $\mathbf{W}_r$ is again a conformable matrix associated with the r^{th} rule. We now explain how to convert $[h]^{\leftarrow}_{rm}$ to an **update vector** $[h]^{\Delta}_{rm}$, and how all update vectors combine to modify $\boxed{h}$.

Discrete-time setting. Here we treat the update vectors $[h]^{\Delta}_{rm}$ as increments to $\boxed{h}$. To update $\boxed{h}$ from time s to time $t = s + 1$, we pool these increments within and across rules (much as in (3)–(4)) and increment by the result:

$$\boxed{h} \mathrel{+}= \sum_r \bigoplus_m^{\beta_r} [h]^{\Delta}_{rm} \quad (9)$$

We skip the update (9) if h has no update vectors. If we apply (9), we first set $\boxed{h}$ to $\mathbf{0}$ if it is null at time s, or has just been set to null at time s by a (2b) rule (docking).

How is $[h]^{\Delta}_{rm}$ obtained? In an ordinary LSTM (Hochreiter & Schmidhuber, 1997), a cell block $\boxed{h}$ is updated by

$$\boxed{h}_{\text{new}} = \mathbf{f} \cdot \boxed{h}_{\text{old}} + \mathbf{i} \cdot (2\mathbf{z} - 1) \quad (10)$$

corresponding to an increment

$$\boxed{h} \mathrel{+}= (\mathbf{f} - 1) \cdot \boxed{h} + \mathbf{i} \cdot (2\mathbf{z} - 1) \quad (11)$$

where the forget gates $\mathbf{f}$, input gates $\mathbf{i}$, and inputs $\mathbf{z}$ are all in $(0, 1)^{D_h}$. Thus, we define $[h]^{\Delta}_{rm}$ as the right side of (11) when $(\mathbf{f}; \mathbf{i}; \mathbf{z}) \stackrel{\text{def}}{=} \sigma([h]^{\leftarrow}_{rm})$, with $[h]^{\leftarrow}_{rm} \in \mathbb{R}^{3D_h}$ from (8).

A small difference from a standard LSTM is that our updated cell values $\boxed{h}$ are transformed into equally many output values $[\![h]\!]$ via equation (7), instead of through $\tanh$ and output gates. A more important difference is that in a standard LSTM, the model's state is a single large cell block. The state update when new input arrives depends on the entire current state. Our innovation is that the update to $\boxed{h}$ (a *portion* of the model state) depends on only a relevant *portion* of the current state, namely $[[\![e]\!]; [\![g_1]\!]; \ldots; [\![g_N]\!]]$. If there are many choices of this portion, (9) pools their effects across instantiations and sums them across rules.

Continuous-time setting. Here we use the continuous-time LSTM as defined by Mei & Eisner (2017), in which cells **drift** between updates to record the passage of time. Each cell drifts according to some parametric function. We will update a cell's parameters just at times when a *relevant* event happens. A fact's embedding $[\![h]\!](t)$ at time t is still

[11] If exogeneous events are used (§2.4), then the instantiations in (8) could include multiple events e that occurred at time s.

given by (7), but $\boxed{h}(t)$ in that equation is given by $\boxed{h}$'s parametric functions as most recently updated (at some earlier time $s < t$). Appendix C reviews the simple family of parametric functions used in the continuous-time LSTM, and specifies how we update the parameters using a collection of update vectors $[h]^{\Delta}_{rm}$ obtained from the $[h]^{\leftarrow}_{rm}$.

Remark. It is common for event atoms `e` to have $D_e = 0$. Then they still have time-varying probabilities (§3.2)—often via `:-` rules whose conditions have time-varying embeddings—but have no embeddings. Even so, different events will result in different updates. This is thanks to Datalog's pattern matching: the event's atom `e` controls which update rules `head <- event, condits...` it triggers, and with what head and condition atoms (since variables in `event` are *reused* elsewhere in the rule). The update to the head atom then depends on the parameters of the selected rules and the current embeddings of their condition atoms.

4. Training and Inference

Suppose we observe that the events on time interval $[0, T]$ are $e_1, \ldots, e_I$ at respective times $t_1 < \cdots < t_I$. In the *continuous-time* setting, the log-likelihood of the parameters is

$$\ell \stackrel{\text{def}}{=} \sum_{i=1}^{I} \log \lambda_{e_i}(t_i) - \int_{t=0}^{T} \lambda(t)\, dt \tag{12}$$

where $\lambda(t) \stackrel{\text{def}}{=} \sum_{e \in \mathcal{E}(t)} \lambda_e(t)$ and $\mathcal{E}(t)$ is the set of event types that are possible at time t. We can estimate the parameters by locally maximizing ℓ using any stochastic gradient method. Details are given in Appendix D, including Monte Carlo approximations to the integral. In the *discrete-time* setting,[12] the integral is replaced by $\sum_{t=1}^{T} \log \lambda(t)$.

Given the learned parameters, we may wish to make a minimum Bayes risk prediction about the next event given the past history. A recipe can be found in Appendix E.

5. Related Work

Past work (Sato, 1995; Poole, 2010; Richardson & Domingos, 2006; Raedt et al., 2007; Bárány et al., 2017) has used logic programs to help define probabilistic relational models (Getoor & Taskar, 2007). These models do not make use of vector-space embeddings or neural networks. Nor do they usually have a temporal component. However, some other (directed) graphical model formalisms do allow the model architecture to be affected by data generated at earlier steps (Minka & Winn, 2008; van de Meent et al., 2018).

Our "neural Datalog through time" framework uses a deductive database augmented with update rules to define and dynamically reconfigure the architecture of a neural generative model. Conditional neural net structure has been used for natural language—e.g., conditioning a neural architecture on a given syntax tree or string (Andreas et al., 2016; Lin et al., 2019). Also relevant are neural architectures that use external read-write memory to achieve coherent sequential generation, i.e., their decisions are conditioned on a possibly symbolic record of data generated from the model at earlier steps (Graves et al., 2014, 2016; Weston et al., 2015; Sukhbaatar et al., 2015; Kumar et al., 2016; Kiddon et al., 2016; Dyer et al., 2016; Lample et al., 2019; Xiao et al., 2019). We generalize some such approaches by providing a logic-based specification language.

Many papers have presented domain-specific sequential neural architectures (Natarajan et al., 2008; Van der Heijden et al., 2014; Shelton & Ciardo, 2014; Meek, 2014; Bhattacharjya et al., 2018; Wang et al., 2019). The models closest to ours are **Know-Evolve** (Trivedi et al., 2017) and **DyRep** (Trivedi et al., 2019), which exploit explicit domain knowledge about how structured events depend on and modify the neural states of their participants. DyRep also conditions event probabilities on a temporal graph encoding binary relations among a fixed set of entities. In §6, we will demonstrate that fairly simple programs in our framework can substantially outperform these strong competitors by leveraging even richer types of knowledge, e.g.: ① Complex n-ary relations among entities that are constructed by join, disjunction, and recursion (§2.1) and have derived embeddings (§2.2). ② Updates to the set of possible events (§2.5). ③ Embeddings of entities and relations that reflect selected past events (§2.4 and §2.6).

6. Experiments

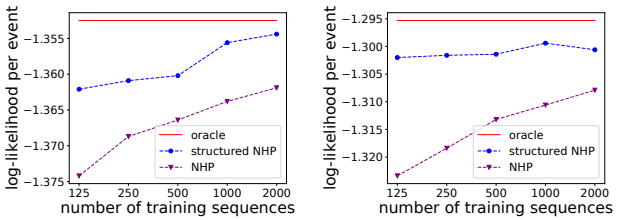
In several continuous-time domains, we exhibit informed models specified using neural Datalog through time (NDTT). We evaluate these models on their held-out log-likelihood, and on their success at predicting the time and type of the next event. We compare with the unrestricted neural Hawkes process (NHP) and with Know-Evolve (KE) and DyRep. Experimental details are given in Appendix F.

We implemented our NDTT framework using PyTorch (Paszke et al., 2017) and pyDatalog (Carbonell et al., 2016). We then used it to implement our individual models—and to reimplement all three baselines, after discussion with their authors, to ensure a controlled comparison. Our code and datasets are available at the URL given in §2.

6.1. Synthetic Superposition Domain

The activities of strangers rarely influence each other, even if they are all observed within a single sequence. We synthesized a domain where each sequence is a superposition of data drawn from M different processes that do not interact with one another at all. Each process generates events of N types, so there are MN total event types `e(M,N)`.

[12] Here each time t has exactly one event (possibly just a `none` event), as the event probabilities sum to 1. So $I = T$ and $t_i = i$.



(a) $M = 8$ (b) $M = 16$

Figure 1. Learning curves of structured model ● and NHP ▼, on sequences drawn from the structured model. The former is significantly better at each training size ($p < 0.01$, paired perm. test).

```
is_process(1).          is_type(1).
      ⋮                       ⋮
is_process(M).          is_type(N).
```

The baseline model is a neural Hawkes process (NHP). It assigns to each event type a separate embedding[13]

```
:- embed(is_event, 8).
is_event(1,1) :- is_process(1), is_type(1).
is_event(1,2) :- is_process(1), is_type(2).
      ⋮
```

This unrestricted model allows all event types to influence one another by depending on and affecting a `world` state:

```
:- event(e, 0).
:- embed(world, 8).
e(M,N) :- world, is_process(M), is_type(N).
world <- init.
world <- e(M,N), is_event(M,N), world.
```

Note that `e(M,N)` in rule 12 has no embedding, since any such embedding would vary along with the probability. As explained in §3.3, rule 12 instead uses `e(M,N)` to draw in the embedding of `is_event(M,N)`, which does not depend on `world` so is static, as called for by the standard NHP.

To obtain a *structured* NHP that recognizes that events from different processes cannot influence each other, we replace `world` with multiple `local` states: each `e(M,N)` only interacts with `local(M)`. Replace rules 9–12 with

```
:- embed(local, 8).
e(M,N) :- local(M), is_type(N).
local(M) <- init, is_process(M).
local(M) <- e(M,N), is_event(M,N), local(M).
```

For various small N and M values (see Appendix F.2), we randomly set the parameters of the structured NHP model and draw training and test sequences from this distribution. We then generated learning curves by training the correclty structured model versus the standard NHP on increasingly long prefixes of the training set, and evaluating them on held-out data. Figure 1 shows that although NHP gradually improves its performance as more training sequences become available, the structured model unsurprisingly learns faster, e.g., only 1/16 as much training data to achieve a higher likelihood. In short, it helps to use domain knowledge of which events come from which processes.

[13]The list of facts like rules 6 and 7 can be replaced by a single rule if we use "parameter names" as explained in Appendix B.

6.2. Real-World Domains: IPTV and RoboCup

IPTV Domain (Xu et al., 2018)**.** This dataset contains records of 1000 users watching 49 TV programs over the first 11 months of 2012. Each event has the form `watch(U,P)`. Given each prefix of the test event sequence, we attempted to predict the next test event's time t, and to predict its program `P` given its actual time t and user `U`.

We exploit two types of structural knowledge in this domain. First, each program `P` has (exactly) 5 out of 22 genre tags such as `action`, `comedy`, `romance`, etc. We encode these as known static facts `has_tag(P,T)`. We allow each tag's embedding ⟦`tag(T)`⟧ to not only influence the embedding of its programs (rule 1) but also track which users have recently watched programs with that tag (rule 2):

```
program(P) :- has_tag(P,T), tag(T).
tag(T) <- watch(U,P), has_tag(P,T).
```

As a result, a program's embedding ⟦`program(P)`⟧ changes over time as its tags shift in meaning.

Second, there is a dynamic hard constraint that a program cannot be watched until it is released, since only then is it added to the database:

```
program(P) <- release(P).
watch(U,P) :- user(U), program(P).
```

Here `release(P)` is an exogenous event with no embedding. More details can be found in Appendix F.3, including full NDTT programs that specify the architectures used by the KE and DyRep papers and by our model.

RoboCup Domain (Chen & Mooney, 2008)**.** This dataset logs actions of soccer players such as `kick(P)` and `pass(P,Q)` during RoboCup Finals 2001–2004. There are 528 event types in total. For each history, we made minimum Bayes risk predictions of the next event's time, and of that event's participant(s) given its time and action type.

Database facts change frequently in this domain. The ball is transferred between robot players at a high rate:

```
!has_ball(P) <- pass(P,Q). % ball passed from P
has_ball(Q)  <- pass(P,Q). % ball passed to Q
```

which leads to highly dynamic constraints on the possible events (since only the ball possessor can `kick` or `pass`):

```
pass(P,Q) :- has_ball(P), teammate(P,Q), ...
```

This example also illustrates how relations between players affect events: the ball can only be `pass`ed to a `teammate`. Similarly, only an `opponent` may `steal` the ball:

```
steal(Q,P) :- has_ball(P), opponent(P,Q), ...
```

We allow each event to update the states of involved players as both KE and DyRep do. We further allow the event observers such as the entire `team` to be affected as well:

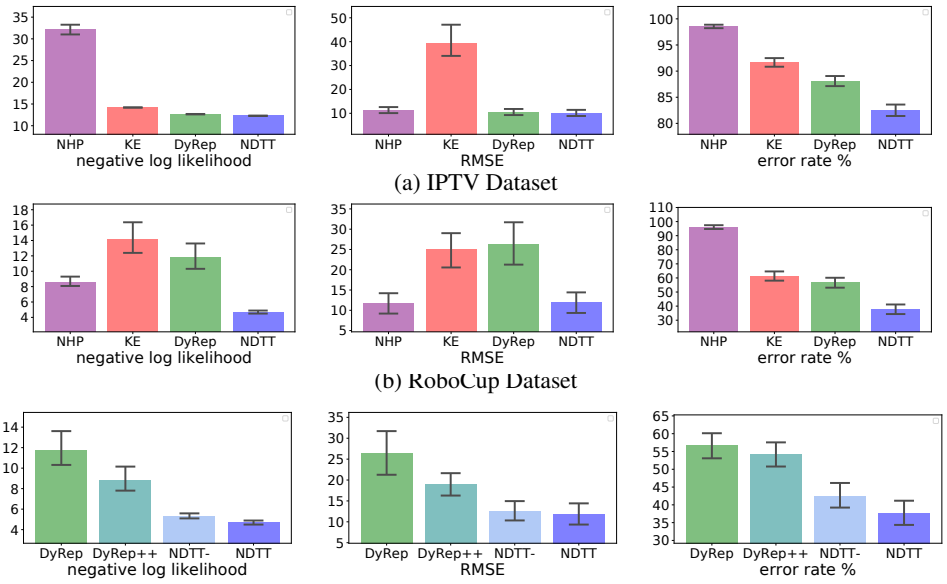


Figure 2. Evaluation results with 95% bootstrap confidence intervals on the real-world datasets of our Datalog program vs. the neural Hawkes process (NHP), KnowEvolve (KE) and DyRep. The RMSE is the root of mean squared error for predicted time. Error rate % denotes the fraction of incorrect predictions of the watched TV program (in IPTV) or the specific player (in RoboCup), given the event time.

Figure 3. Ablation study in the RoboCup domain. "DyRep++" has the same <- rules as our structured model and "NDTT−" uses 0-dimensional `team` embeddings.

```
team(T) <- pass(P,Q), in_team(P,T), ....
```

so all players can be aware of this event by consulting their `team` states. More details can be found in Appendix F.5, including our full Datalog programs. The hard logical constraints on possible events are not found in past models.

Results and Analysis. After training, we used minimum Bayes risk (§4) to predict events in test data (details in Appendix E). Figure 2 shows that our NDTT model enjoys consistently lower error than strong competitors, across datasets and prediction tasks.

NHP performs poorly in general since it doesn't consider any knowledge. KE handles relational information, but doesn't accommodate dynamic facts such as `released(game_of_thrones)` and `has_ball(a8)` that reconfigure model architectures on the fly.

In the IPTV domain, DyRep handles dynamic facts (e.g., newly released programs) and thus substantially outperforms KE. Our NDTT model's moderate further improvement results from its richer :- and <- rules related to `tag`s.

In the RoboCup domain, our reimplementation of DyRep allows deletion of facts (player losing ball possession), whereas the original DyRep only allowed addition of facts. Even with this improvement, it performs much worse than our full NDTT model. To understand why, we carried out further ablation studies, finding that NDTT benefits from its hybridization of logic and neural networks.

Ablation Study I: Taking Away Logic. In the RoboCup domain, we investigated how the model performance degrades if we remove each kind of rule from the NDTT model. We obtained "NDTT−" by dropping the `team` states, and "DyRep++" by not tracking the ball possessor. The latter is still an enhancement to DyRep because it adds useful <- rules: the first "+" stands for the <- rules in which some conditions are not neighbors of the head, and the second "+" stands for the <- rules that update event observers.

As Figure 3 shows, both ablated models outperform DyRep but underperform our full NDTT model. DyRep++ is interestingly close to NDTT on the participant prediction, implying that its neural states learn to track who possesses the ball—though such knowledge is not tracked in the logical database—thanks to rich <- rules that see past events.

Ablation Study II: Taking Away Neural Networks. We also investigated how the performance of our structured model would change if we reduce the dimension of all embeddings to zero. The model still knows logically which events are possible, but events of the same type are now more interchangeable. The performance turns out to degrade greatly, indicating that the neural networks had been learning representations that are actually helpful for prediction. See Appendix F.8 for discussion and experiments.

7. Conclusion

We showed how to specify a neural-symbolic probabilistic model simply by writing down the rules of a deductive database. "Neural Datalog" makes it simple to define a large set of structured objects ("facts") and equip them with embeddings and probabilities, using pattern-matching rules to explicitly specify which objects depend on one another.

To handle temporal data, we proposed an extended notation to support *temporal* deductive databases. "Neural Datalog through time" allows the facts, embeddings, and probabilities to change over time, both by gradual drift and in response to discrete events. We demonstrated the effectiveness of our framework by generatively modeling irregularly spaced event sequences in real-world domains.

Acknowledgments

We are grateful to Bloomberg L.P. for enabling this work through a Ph.D. Fellowship Award to the first author, and to the National Science Foundation for supporting the other JHU authors under Grant No. 1718846. We thank Karan Uppal, Songyun Duan and Yujie Zha from Bloomberg L.P. for helpful comments and support to apply the framework to Bloomberg's real-world data. We thank the anonymous ICLR reviewers for helpful comments on an earlier version of this paper, Hongteng Xu for such comments and also for additional data, and Rakshit Trivedi for insightful discussion about Know-Evolve and DyRep. Moreover, we thank NVIDIA Corporation for kindly donating two Titan X Pascal GPUs, and the state of Maryland for the Maryland Advanced Research Computing Center.

References

Acar, U. A. and Ley-Wild, R. Self-adjusting computation with Delta ML. In *International School on Advanced Functional Programming*, 2008.

Aldous, D., Ibragimov, I., Jacod, J., and Aldous, D. Exchangeability and related topics. In *École d'Été de Probabilités de Saint-Flour XIII — 1983*, Lecture Notes in Mathematics. 1985.

Andreas, J., Rohrbach, M., Darrell, T., and Klein, D. Learning to compose neural networks for question answering. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics Human Language Technologies (NAACL HLT)*, 2016.

Bárány, V., ten Cate, B., Kimelfeld, B., Olteanu, D., and Vagena, Z. Declarative probabilistic programming with Datalog. *ACM Transactions on Database Systems*, 42 (4):22:1–35, October 2017.

Bhattacharjya, D., Subramanian, D., and Gao, T. Proximal graphical event models. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 8136–8145, 2018.

Blei, D. and Lafferty, J. Correlated topic models. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 18, pp. 147–154, 2006.

Blei, D. M. and Frazier, P. I. Distance-dependent Chinese restaurant processes. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 87–94, 2010.

Carbonell, P., `jcdouet`, Alves, H. C., and Tim, A. pyDatalog, 2016.

Ceri, S., Gottlob, G., and Tanca, L. What you always wanted to know about Datalog (and never dared to ask). *IEEE Transactions on Knowledge and Data Engineering*, 1989.

Chen, D. L. and Mooney, R. J. Learning to sportscast: A test of grounded language acquisition. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2008.

Dyer, C., Kuncoro, A., Ballesteros, M., and Smith, N. A. Recurrent neural network grammars. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics Human Language Technologies (NAACL HLT)*, 2016.

Elman, J. L. Finding structure in time. *Cognitive Science*, 1990.

Filardo, N. W. and Eisner, J. A flexible solver for finite arithmetic circuits. In *Technical Communications of the 28th International Conference on Logic Programming (ICLP)*, 2012.

Fisher, R. A., Corbet, A. S., and Williams, C. The relation between the number of species and the number of individuals in a random sample of an animal population. *J. Animal Ecology*, 12:42–58, 1943.

Getoor, L. and Taskar, B. (eds.). *Introduction to Statistical Relational Learning*. MIT Press, 2007.

Goller, C. and Kuchler, A. Learning task-dependent distributed representations by backpropagation through structure. In *IEEE International Conference on Neural Networks*, volume 1, pp. 347–352, 1996.

Graves, A., Wayne, G., and Danihelka, I. Neural Turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

Graves, A., Wayne, G., Reynolds, M., Harley, T., Danihelka, I., Grabska-Barwińska, A., Colmenarejo, S. G., Grefenstette, E., Ramalho, T., Agapiou, J., et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 2016.

Hamilton, W., Ying, Z., and Leskovec, J. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017a.

Hamilton, W. L., Ying, R., and Leskovec, J. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*, 2017b.

Hammer, M. A. *Self-Adjusting Machines*. PhD thesis, Computer Science Department, University of Chicago, 2012.

Hawkes, A. G. Spectra of some self-exciting and mutually exciting point processes. *Biometrika*, 1971.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural Computation*, 1997.

Kiddon, C., Zettlemoyer, L., and Choi, Y. Globally coherent text generation with neural checklist models. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2016.

Kingma, D. and Ba, J. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

Kumar, A., Irsoy, O., Ondruska, P., Iyyer, M., Bradbury, J., Gulrajani, I., Zhong, V., Paulus, R., and Socher, R. Ask me anything: Dynamic memory networks for natural language processing. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2016.

Lample, G., Sablayrolles, A., Ranzato, M., Denoyer, L., and Jégou, H. Large memory layers with product keys. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Le, P. and Zuidema, W. The forest convolutional network: Compositional distributional semantics with a neural chart and without binarization. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2015.

Lin, C., Zhu, H., Gormley, M. R., and Eisner, J. M. Neural finite-state transducers: Beyond rational relations. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics Human Language Technologies (NAACL HLT)*, 2019.

Lin, C.-C. and Eisner, J. Neural particle smoothing for sampling from conditional sequence models. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics Human Language Technologies (NAACL HLT)*, 2018.

Ling, W., Luís, T., Marujo, L., Astudillo, R. F., Amir, S., Dyer, C., Black, A. W., and Trancoso, I. Finding function in form: Compositional character models for open vocabulary word representation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2015.

Liniger, T. J. *Multivariate Hawkes processes*. Diss., Eidgenössische Technische Hochschule ETH Zürich, Nr. 18403, 2009.

Meek, C. Toward learning graphical and causal process models. In *Uncertainty in Artificial Intelligence Workshop on Causal Inference: Learning and Prediction*, volume 1274, pp. 43–48, 2014.

Mei, H. and Eisner, J. The neural Hawkes process: A neurally self-modulating multivariate point process. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Mei, H., Qin, G., and Eisner, J. Imputing missing events in continuous-time event streams. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2019.

Mikolov, T., Karafiát, M., Burget, L., Cernocký, J., and Khudanpur, S. Recurrent neural network-based language model. In *Proceedings of the Annual Conference of the International Speech Communication Association (INTERSPEECH)*, 2010.

Minka, T. and Winn, J. Gates: A graphical notation for mixture models. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 1073–1080, 2008.

Natarajan, S., Bui, H. H., Tadepalli, P., Kersting, K., and Wong, W.-K. Logical hierarchical hidden Markov models for modeling user activities. In *Proceedings of the International Conference on Inductive Logic Programming (ICILP)*, 2008.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. Automatic differentiation in PyTorch. 2017.

Poole, D. AILog user manual, version 2.3, 2010.

Raedt, L. D., Kimmig, A., and Toivonen, H. Problog: A probabilistic Prolog and its application in link discovery. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2007.

Richardson, M. and Domingos, P. Markov logic networks. *Machine Learning*, 2006.

Sato, T. A statistical learning method for logic programs with distribution semantics. In *Proceedings of the International Conference on Logic Programming (ICLP)*, pp. 715–729, 1995.

Shelton, C. R. and Ciardo, G. Tutorial on structured continuous-time Markov processes. *Journal of Artificial Intelligence Research*, 51:725–778, 2014.

Socher, R., Huval, B., Manning, C. D., and Ng, A. Y. Semantic compositionality through recursive matrix-vector spaces. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2012.

Srivastava, R. K., Greff, K., and Schmidhuber, J. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

Sukhbaatar, S., Weston, J., Fergus, R., et al. End-to-end memory networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2015.

Sundermeyer, M., Ney, H., and Schluter, R. LSTM neural networks for language modeling. In *Proceedings of the Annual Conference of the International Speech Communication Association (INTERSPEECH)*, 2012.

Swift, T. and Warren, D. S. XSB: Extending Prolog with tabled logic programming. *Theory and Practice of Logic Programming*, 12(1–2):157–187, 2012.

Tai, K. S., Socher, R., and Manning, C. D. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2015.

Tran, K. M., Bisk, Y., Vaswani, A., Marcu, D., and Knight, K. Unsupervised neural hidden Markov models. In *Proceedings of the Workshop on Structured Prediction for NLP*, pp. 63–71, Austin, TX, November 2016.

Trivedi, R., Dai, H., Wang, Y., and Song, L. Know-Evolve: Deep temporal reasoning for dynamic knowledge graphs. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2017.

Trivedi, R., Farajtabar, M., Biswal, P., and Zha, H. DyRep: Learning representations over dynamic graphs. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

van de Meent, J.-W., Paige, B., Yang, H., and Wood, F. An introduction to probabilistic programming. *arXiv preprint arXiv:1809.10756*, 2018.

Van der Heijden, M., Velikova, M., and Lucas, P. J. Learning Bayesian networks for clinical time series analysis. *Journal of Biomedical Informatics*, 2014.

Wang, Y., Smola, A., Maddix, D. C., Gasthaus, J., Foster, D., and Januschowski, T. Deep factors for forecasting. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2019.

Weston, J., Chopra, S., and Bordes, A. Memory networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

Williams, R. J. and Zipser, D. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280, 1989.

Xiao, C., Teichmann, C., and Arkoudas, K. Grammatical sequence prediction for real-time neural semantic parsing. In *Proceedings of the ACL Workshop on Deep Learning and Formal Languages: Building Bridges*, 2019.

Xu, D., Ruan, C., Korpeoglu, E., Kumar, S., and Achan, K. Inductive representation learning on temporal graphs. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

Xu, H., Luo, D., and Carin, L. Online continuous-time tensor factorization based on pairwise interactive point processes. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2018.

Zhang, X., Lu, L., and Lapata, M. Top-down tree long short-term memory networks. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics Human Language Technologies (NAACL HLT)*, 2016.