Deep Neural Networks for Approximating Stream Reasoning with C-SPARQL

Ricardo Ferreira, Carolina Lopes, Ricardo Gonçalves, Matthias Knorr, Ludwig Krippahl, and João Leite

NOVA LINCS & Departamento de Informática, Universidade Nova de Lisboa

Abstract. The amount of information produced, whether by newspapers, blogs and social networks, or by monitoring systems, is increasing rapidly. Processing all this data in real-time, while taking into consideration advanced knowledge about the problem domain, is challenging, but required in scenarios where assessing potential risks in a timely fashion is critical. C-SPARQL, a language for continuous queries over streams of RDF data, is one of the more prominent approaches in stream reasoning that provides such continuous inference capabilities over dynamic data that go beyond mere stream processing. However, it has been shown that, in the presence of huge amounts of data, C-SPARQL may not be able to answer queries in time, in particular when the frequency of incoming data is higher than the time required for reasoning with that data. In this paper, we investigate whether reasoning with C-SPARQL can be approximated using Recurrent Neural Networks and Convolutional Neural Networks, two neural network architectures that have been shown to be well-suited for time series forecasting and time series classification, to leverage on their higher processing speed once the network has been trained. We consider a variety of different kinds of queries and obtain overall positive results with high accuracies while improving processing time often by several orders of magnitude.

1 Introduction

Large amounts of data are constantly being produced, whether by newspapers, blogs and social networks, or by monitoring systems (such as traffic sensors, financial market prediction, weather forecasting, etc.) [22]. For such data streams, it is often necessary to be able to infer new information with high efficiency in real time. For example, observing data about a patient's health status, diet and physical activity can help to anticipate health problems and request medical support in case of an emergency. Also monitoring traffic in a city area can help react to problems, such as traffic jams or accidents, in real time allowing to reroute traffic to improve travel time and reduce environmental impact.

Data Stream Management Systems (DSMS) and Complex Event Processors (CEP) tackle this problem [7], where the former allow continuous querying over a data stream and the latter aim at identifying patterns of events that occur in a data stream. However, these systems cannot handle situations when the data

is heterogeneous and it is necessary to integrate background knowledge, such as the patient's record, data on medication, or general medical knowledge expressed in an ontology, and perform more complex reasoning tasks.

Stream reasoning aims at overcoming these limitations [8,9,22] and one of the more prominent approaches is C-SPARQL (Continuous SPARQL) [1,2], a language for continuously querying over streams, that combines the features of DSMS and CEP, and the ability to incorporate background knowledge. C-SPARQL builds on language standards for the Semantic Web whose development has been driven by the World Wide Web Consortium (W3C), namely, on the Resource Description Framework (RDF) [23], a standard model of data exchange on the Web which has led to the development of Linked Open Data [16] with a large amount of structured and interconnected data,[1] and SPARQL the query language for querying over RDF data [14]. In more detail, C-SPARQL is able to process queries over various RDF streams simultaneously, employing so-called windows that focus only on a limited (recent) portion of the stream, taking into account background knowledge in the form of RDF graphs. Query answers can be variable bindings or again RDF graphs, and both of these even in the form of streams, thus C-SPARQL is capable of updating knowledge bases as new information arrives.

While C-SPARQL is thus in principle well-suited to perform stream reasoning, these advanced reasoning capabilities come at a price [27]. It has been shown that there is a limit on the amount of triples the system is capable of processing per second, which varies depending on the complexity of the considered query, and which may be prohibitive in real world scenarios. In fact, at higher rates where the amount of triples per second is superior than the limit, C-SPARQL provides erroneous answers.

It has been argued that in the face of huge amounts of data, sound and complete reasoning can be considered as a gold standard, but for obtaining answers in a timely fashion, approximate methods need to be applied [18]. These approximate methods include approaches based on machine learning, such as Neural Networks (NNs) [15], which are able to learn and generalize patterns from a given data set, making them, once trained, applicable to unseen situations with a considerably higher processing speed and robust to noisy data [17]. These methods have gained further interest with the advent of Deep Neural Networks [13], which are behind a variety of substantial recent advances in Artificial Intelligence, for example, in speech and visual recognition oe vehicle control, and which allow to detect considerably more sophisticated patterns in a data set.

A few solutions to such deductive reasoning using Deep Learning have appeared within the field of neural-symbolic integration [3,17]. Namely, Makni and Hendler [21] propose noise-tolerant reasoning for RDF(S) knowledge graphs [5], Hohenecker and Lukasiewicz [20] introduce Recursive Reasoning Networks for OWL 2 RL reasoning [24], and Ebrahimi et al. [10] tackle learning of simple deductive RDFS reasoning. However, none of these approaches takes streaming

---

[1] https://lod-cloud.net/

data into account, making them unsuitable in the scenarios where reacting to temporal sequences of events is required.

In this paper, we investigate whether reasoning with C-SPARQL can be approximated using Deep Learning. We consider Recurrent Neural Networks (RNNs) [26] and Convolutional Neural Networks (CNN) [28], two neural network architectures that have been shown to be well-suited for time series forecasting and time series classification [4,11]. Using a data set containing sensor data on traffic, pollution and weather conditions, we consider different kinds of queries aiming to cover different features within the expressiveness C-SPARQL offers, and we generate the target labels for the training set using C-SPARQL itself to avoid the cost of manually labeling the data. We are able to show that such approximate reasoning is indeed possible and obtain overall positive results with high accuracies while improving processing time often by several orders of magnitude. We also provide considerations on which of the two arquitectures is more suitable in which situation.

2   Background

In this section, we recall relevant notions to facilitate the reading of the remaining material, namely on C-SPARQL, and the two kinds of neural networks we consider, recurrent and convolutional neural networks.

2.1   C-SPARQL

C-SPARQL (Continuous SPARQL) [1,2] is a declarative query language that combines the features of DSMS and CEP to continuously query over streams of data taking into account background knowledge. C-SPARQL builds on the Resource Description Framework (RDF) [23], a standard model of data exchange on the Web and SPARQL the query language for querying over RDF data [14].

To represent continuous streams of data, RDF streams are introduced [1]. An *RDF stream* is an ordered sequence of pairs, where each pair consists of an RDF triple $\langle subject, predicate, object \rangle$ and its timestamp τ. Subsequent timestamps τ_i and τ_{i+1} are monotonically non-decreasing ($\tau_i \leq \tau_{i+1}$), i.e., any (unbounded, though finite) number of consecutive triples can have the same timestamp, but they still occur sequentially according to the given order.

To be able to process such streaming data, C-SPARQL applies *windows* to delimit a finite amount of triples to be considered, which aligns with the idea that we cannot consider the entire stream when reasoning (nor store it). Such windows can be *physical*, i.e., a specific number of triples is selected, or *logical*, i.e., the triples that occur in a certain interval of time are selected. The latter can be *sliding* windows, when they are progressively advancing with a time step smaller than their interval, or *non-overlapping* (also called *tumbling*) when they are advancing with exactly their time interval at each iteration [2].

Continuous queries in C-SPARQL extend SPARQL queries [14] with the necessary features to handle streams of data. More concisely, a C-SPARQL query

starts with a registration statement to be able to produce continuous output in the form of variable bindings, tables or graphs, also indicating at which frequency the query is processed. A basic C-SPARQL query then contains *SELECT*, *FROM* and *WHERE* statements, where the *SELECT* statement indicates the variables one is interested in, the *FROM* statement the IRI of the stream considered, including the definition of the kind of applied window, and the *WHERE* statement a condition for the query. In addition, C-SPARQL queries also permit the usage of the following advanced characteristics:

- Aggregate functions, namely, count, sum, average, minimum and maximum, which additionally allow grouping results as well as subsequent filters on the aggregated data;
- Incorporation of static background knowledge in the form of external RDF documents that can be referred to in the query (optionally introduced in the prefix statement);
- Usage of the timestamp function to be able to compare the time of occurrence of different events in the window of the stream;
- Querying various streams simultaneously;
- Stream the results of the continuous query, in addition to returning the variable bindings and graphs.

For the concise description of the syntax of SPARQL queries, and their formal semantics based on mappings we refer to [2].

It should be noted that, as shown in [27], the query execution time depends on the complexity of the query and increases linearly with the growth of the window size and the size of the static background knowledge, effectively limiting the number of triples that can be processed per unit of time, resulting in wrong answers if this threshold is passed.

2.2 Neural Networks for Time Series Classification

We assume a basic understanding of (deep) neural networks and how they can be trained using supervised learning [13,15]. Here, we provide an overview on the two network architechtures particularly well-suited for time series classification.

Recurrent Neural Networks Recurrent neural networks (RNNs) are a type of deep neural networks specially designed for sequence modelling, that have received a great amount of attention due to their flexibility in capturing nonlinear relationships [26]. An RNN is very similar to a feedforward neural network with the exception that, in RNNs, the output of the recurrent layer is passed as input to that same layer in the next time step. At each time step, the neurons receive an input vector and the output vector from the previous time step. RNNs have had great success in forecasting and classifying time series. However, they suffer from the vanishing gradient problem [25], which causes the RNN to lose the ability to "learn more" at a certain point and results in difficulties capturing long-term dependencies. To overcome both problems, more sophisticated architectures were

introduced such as Long Short-Term Memory Units (LSTM) [19], and Gated Recurrent Units (GRU) [6], which are a simplified version of LSTM, aimed at being useful for smaller datasets.

Convolutional Neural Networks Convolutional Neural Networks (CNNs) have first been applied very successfully in the context of image recognition, and have shown impressive results when dealing with Time Series Classification [11]. Their distinguishing characteristic is the usage of convolutional and pooling layers for detecting patterns [28].

A convolutional layer aims at recognizing patterns that exist in the data. This layer applies a finite number of filters to the input, creating as output, for each filter, a feature map with the patterns detected by that filter. The values of the kernel matrix used in the convolution operation for each filter are adjusted during training, so that the network can learn the patterns it needs to find. Pooling layers then allow us to reduce the dimensions of such data by joining the outputs from a portion of the neurons in the previous layer into a single neuron of the following layer, using, e.g., *Max pooling* where the highest value is selected. This way, the data is compressed, simplifying the result without losing the most relevant information. Combining these layers leads to a gradual reduction in the amount of information to be processed along the network and requires fewer parameters than an equivalent fully connected network.

3 Methodology

In order to test our hypothesis that stream reasoning with C-SPARQL can be approximated using RNNs and CNNs, we designed and executed a series of experiments, whose rationale is explained in this section.

In general, each experiment consists of performing the following two steps.

1. Formulate a C-SPARQL query using different combinations of features (mentioned in Section 2) and execute this query using C-SPARQL to obtain the correct query answers;
2. Use an encoding of the original data together with the stream reasoner's answers for developing, training, and testing various models for RNNs and CNNs to determine, for each of them, the one with the best performance in terms of approximation of reasoning.

Both steps are detailed next after introducing the test data we have used.

3.1 Dataset

To perform our experiments, we chose a publicly available dataset [2] that gathers data on traffic, pollution and weather conditions by 449 sensors distributed

---

[2] `http://iot.ee.surrey.ac.uk:8080/datasets.html`

throughout the city of Aarhus in Denmark. The data collected by the sensors is diverse, including average speed, number of vehicles detected, measurements of, for example, carbon, sulfur, ozone, nitrogen, as well as data on temperature, pressure and humidity.

To aid our experiments, we created an event processor that returns discretized events from the data in the dataset. The benefits of this are two-fold. Given the known limitations in terms of capacity of processing large amounts of data of C-SPARQL, we can perform simple event detection a-priori to reduce the amount of data to be processed with C-SPARQL and at the same time avoid that it be applied for simple event detection where it would not be necessary in the first place. Such pre-processing also facilitates encoding the streaming data as input for the neural networks, and, thus, in addition allows for a fairer comparison of processing time, as both use the same pre-processed set of input data.

This event processor is a Python script[3] that creates certain events based on the present data, namely *movement, no_movement, normal air, low carbon, high carbon, low nitrogen, high nitrogen, low sulfure* and *high sulfure*. For example, if a sensor does not detect any vehicles at a given time instant, the event detector would return the *no_movement* event, and if the carbon values are above normal (based on average values), the *high_carbon* event. In addition, to allow for more interesting queries involving advanced reasoning, we divided the city (and thus the sensors) into 10 sectors with the aim to allow comparisons between different areas. To balance the amount of data per sector, we only considered 15 sensors per sector, and condensed the time step in the original dataset from 5 minutes to 1 minute. Thus, all computed events are associated with their sector and a timestamp, given in minutes. The dataset resulting from this process contains 17532 samples of sliding windows (with a time step of 1 minute) of size 5 corresponding to 5 minutes.

3.2 C-SPARQL Queries

The queries we designed aim to leverage the expressive features C-SPARQL offers. In our experiments, we thus created queries of varying complexity using different combinations of the features, most notably aggregations, background knowledge and time comparisons, incorporating combinatorics over the different sectors, which is not easily coverable by a stream processor.

Considering that Neural Networks are not able to straightforwardly provide RDF triples as answers, we only took into account queries that return answers that can be encoded into a constant number of neurons, i.e., queries that return a Boolean answer (ASK operator) or a fixed number of answers. For similar reasons, we also abstained from considering C-SPARQL queries that return a stream of RDF data. This is not a major limitation and the queries we present in Section 4 are expressive and showcase a wide variety of possible use cases.

To process C-SPARQL queries we have used the Ready-To-Go Pack.[4] When processing a query, time and memory usage were measured. To avoid that

---

[3] `https://github.com/CarolinaMagLopes/Deep-Neural-Networks-for-C-SPARQL`.

[4] `http://streamreasoning.org/resources/c-sparql`

C-SPARQL starts giving wrong answers when passing the threshold of processable amount of data per time step, we added sufficient delay to the query processing (as determined in prior tests and higher than any processing times), without affecting the measured processing times themselves.

3.3 Training RNNs and CNNs

The result of running the queries are windows of data together with the corresponding query result. To be able to train a neural network with this data, we needed to encode the data accordingly. To cover the 9 events in 10 sectors, a matrix of size (9×10) has been used. For the case of RNNs, a window of 5 minutes thus results in an input of 5 such matrices, whereas for CNNs an additional dimension of 5 is added, hence each window corresponds to a matrix of size ($9 \times 10 \times 5$).

Then for each query, varying architectures with different numbers (and kinds) of layers and neurons were designed, trained, and tested for both CNNs as well as LSTM and GRU architectures, the RNNs, we considered here, using also Dropout and Gaussian Noise to avoid overfitting of the networks. The quality of their reasoning approximation has been measured using the accuracy (the fraction of correct answers) obtained with the test set and from training.

From the 17532 samples, 1532 were reserved for testing and 16000 were used for training and validation, typically with 10% of these being used in the validation set. The number of epochs used for training varies depending on the used network and the query, corresponding in all cases to the best results achieved while avoiding overfitting. For RNNs, this varies between 100 and 1000 epochs, for CNN, 50 epochs provided the best results.

The networks were developed and tested using Python3, namely using *TensorFlow*[5], *Keras*[6] and *SciKit-Learn* [7], where Keras in particular provides implementations of CNNs, LSTM and GRU architectures.

4 Experiments and Results

In this section, we present the different experiments carried out with the aim of assessing to what extent neural networks are able to approximate reasoning with C-SPARQL. Among the many different queries that we tested, we have chosen several representatives of certain combinations of the features of C-SPARQL (cf. Section 2), and grouped them together in subsections according to the chosen features to facilitate the reading.[8]

In each case, we indicate the representative queries and the processing times of C-SPARQL, RNNs and CNNs, as well as the resulting accuracy for the test

---

[5] https://www.tensorflow.org/

[6] https://keras.io/

[7] https://scikit-learn.org/stable/

[8] An extended version of the paper contains the exact encoding of the queries, final configurations of the networks, and plots of the learning phase [12].

set for both network architectures. Please note that w.r.t. RNNs, we often refer directly to LSTM in this section, as they turned out to provide better results than GRU for all the tests of RNNs.

The experiments were run on a computer with an Intel Core i3-3240T processor with 2.90GHz and 4GB of RAM.

As this is common to all experiments, we remark here that running the event processor to create the data set of events only required on average a few milliseconds per sample, meaning that this is irrelevant for the overall processing time of windows covering 5 minutes, i.e., it is insignificant for ensuring whether processing queries is possible in real-time. We also note that our experiments when processing with C-SPARQL have shown overall that, even for more complicated queries, memory consumption does nor surpass a few hundred MB, which does not constitute a bottle-neck for processing or running the tests, which is why we do not report it individually. Finally, we also report here that training the networks took on average between 45 minutes to 4.5 hours for RNNs and 4 to 10 minutes for CNNs, depending on the query and the number of epochs to achieve the best results. While this adds to the time necessary to use a neural network instead of C-SPARQL, it only needs to be done once before applying the network for approximating reasoning. Hence, once the network is trained, this does not affect the usage in real time either.

4.1 Queries with temporal events

For the first set of queries, we tested the identification of sequences of temporal events for RDF triples with the timestamp function, which is one of the fundamental characteristics of C-SPARQL.

Here, we wanted to determine within a window the occurrence, in a sector, of the complex event composed of

t_1: *normal_air* & *no_movement* t_2: *high_carbon* & *movement*
t_3: *high_sulfure*

where t_1, t_2 and t_3 are timestamps such that $t_1 < t_2 < t_3$.

For this event, we created different kinds of queries to cover the expressiveness of C-SPARQL. Query 1 selects the sectors where the complex event occurred (using the SELECT operator), while Query 2, tests whether the event occurred in all ten sectors (using the ASK operator). Query 3 is considerably more complex than the previous two since aggregation is added in the form of COUNT and MAX, aiming to determine the sector with the highest number of occurrences of this complex event. The obtained results are reported in the following table.[9]

Results	C-SPARQL Time	LSTM Train Acc	Test Acc	Time	CNN Train Acc	Test Acc	Time
Query 1	10-25min	0.9801	0.9780	$180\mu s$	0.9832	0.9976	$280\mu s$
Query 2	10-15min	0.8760	0.8695	$210\mu s$	0.9898	0.9852	$450\mu s$
Query 3	15-30+min	0.9350	0.9311	$700\mu s$	0.9324	0.9222	$240\mu s$

---

[9] C-SPARQL processing times vary depending on the number of tuples per window.

We can see that CNNs provide excellent results for both Queries 1 and 2, whereas LSTMs provide excellent results as well for Query 1, but only good results for Query 2 (ASK), which is already observed during training. Such difficulties can be circumvented though by counting the results of Query 1, for which the results are excellent and obtain the correct answer with high precision. For Query 3, which is considerably more complex with the aggregations, very good results are achieved for both kinds of networks.

Overall, we conclude that detecting temporal sequences in stream reasoning using neural networks is feasible with, in general, very good results even for more complex queries. At the same time, processing with any of the trained networks is at least 6 orders of magnitudes faster than reasoning with C-SPARQL. In particular, for the queries considered here, C-SPARQL could not be used with the considered data in real time as the processing time is far higher than the admitted processing time of 1 minute (each window captures 5 minutes, but the time step of the sliding window is 1 minute), thus resulting in incorrect answers in such a setting, unlike the networks that easily permit processing in real time.

4.2 Queries with Background Knowledge

In the second part, we introduced an additional layer of complexity by adding background knowledge in the form of an ontology containing information about the type of sectors ($school\_area$, $urban\_area$, etc.), events, called infractions, not allowed in the various types of sectors (for example, the $high\_carbon$ event should not happen in $school\_area$) and information about adjacency between sectors. Here, we did not consider temporal comparisons between events, but made use of various aggregation functions.

Query 4 consists in selecting the sectors where more infractions occurred than the average of infractions in the adjacent sectors, thus making use of the topological knowledge, whereas Query 5 asks if the number of sectors, where more infractions occurred than the average of infractions in the adjacent sectors, is greater than or equal to 4, requiring aggregations within an aggregation.

For Query 6, we further increased the complexity of the query and made use of the Property Paths Operators that C-SPARQL provides, namely, among them the '/' operator, where $pred_1/pred_2$ corresponds to the sequence path of $pred_1$ followed by $pred_2$. This allows to query for sectors where more infractions occurred than the average of infractions in the sectors adjacent to adjacent sectors. To allow for more meaningful answers, for this query, we increased the number of sectors to 15 choosing sensors in the same way as described in Section 3, since with the initial 10 sectors, any sector would be close to almost all the others.

	C-SPARQL	LSTM			CNN		
Results	Time	Train Acc	Test Acc	Time	Train Acc	Test Acc	Time
Query 4	20-40ms	0.9810	0.9775	1ms	0.9797	0.9799	$545\mu s$
Query 5	20ms	0.9130	0.8930	1ms	0.9180	0.9066	$460\mu s$
Query 6	30ms	0.9840	0.9771	1ms	0.9795	0.9728	$700\mu s$

Both LSTMs and CNNs perform similarly, showing excellent results for both queries 4 and 6, and only good results for Query 5 (using ASK). Similar to the

solution for Query 2 for LSTMs, we can circumvent this and take advantage of the excellent performance for Query 4 and simply count the results there. We observe that C-SPARQL reasoning involving ontologies can be captured with LSTMs and CNNs as well. Notably, for this kind of queries C-SPARQL provides very good processing times that allow its usage in real time, but using the trained networks is still at least 30 and 60 times faster for LSTMs and CNNs, resp.

4.3 Combining Temporal Events and Background Knowledge

In the final set of queries, we combine comparisons of temporal events with background knowledge and aggregations to obtain highly sophisticated queries for testing our hypothesis. We reuse the complex event from Section 4.1 and the ontology from Section 4.2 for the two final queries. Namely, with Query 7 we determine those sectors that have more occurrences of the complex event than the average in the adjacent sectors, and with Query 8, similar to Query 6, those sectors that have more occurrences of the complex event than in the close, but not immediately adjacent sectors.

	C-SPARQL	LSTM			CNN		
Results	Time	Train Acc	Test Acc	Time	Train Acc	Test Acc	Time
Query 7	20-30+min	0.8990	0.8864	1ms	0.8310	0.8317	$460\mu s$
Query 8	25-30+min	0.8860	0.8718	1ms	0.8584	0.8555	$555\mu s$

We can observe that both kinds of networks achieve good results. Given the observed results also from training, where our tests with more advanced models would result in overfitting, we believe that this may be due to the fact that for such complicated queries more training data would be necessary, and leave this for future work. We also note that, here, LSTMs show a slightly better performance. We conjecture that this could be related to the fact that for a CNN the entire input is presented as a matrix possibly somewhat obfuscating the temporal aspect, whereas for LSTMs the temporal component of the input is separated, and thus possibly easier to distinguish. In any case, similar to Section 4.1, C-SPARQL cannot be applied in real time, as the processing time by far exceeds the limit, whereas the networks are again six orders of magnitude faster. Thus, even though there is still space for improvement here in terms of resulting precision for the approximation, given the unsuitability of using C-SPARQL in real time for such queries, our approach is also very promising for such highly complex queries.

5   Conclusions

We have investigated whether expressive stream reasoning with C-SPARQL can be approximated using Recurrent and Convolutional Neural Networks, since for more sophisticated queries with higher quantities of data to be processed, C-SPARQL does not process the data fast enough to provide timely answers.

Our experiments on a real data set containing among others data on traffic and air pollution show that both RNNs and CNNs are well-suited for this task, as their processing time is vastly superior compared to C-SPARQL, in particular when relative comparisons of temporal events are required, and in many cases, they show excellent or very good results of approximation of reasoning, even when utilizing more complex combinations of C-SPARQL's features.

When comparing both architectures, we noted slightly better results for RNNs ,i.e., LSTMs, when approximating highly complex queries in particular including many temporal comparisons (Query 7 and 8). On the other hand, CNNs performed a bit better for ASK queries (notably Query 2), which requires counting globally, and training them is notably faster. Our preliminary conclusion is that CNNs could be more suitable where space perception is more important, while RNNs seem preferable for queries more oriented towards temporal aspects.

Still, C-SPARQL remains more suitable if the frequency of incoming data is below its processing capabilities and when obtaining the correct result is mandatory, or where ad-hoc variants of a query are needed, which would require re-training the network, as well as in situations when returning answer substitutions where the domain is not previously limited and known, or when returning constructed graphs (as streams).

In terms of future work, it would be interesting to further explore the situations where the networks were not providing excellent results. This could be tackled by trying to use larger datasets and further test the combinations of features of C-SPARQL that proved more difficult when approximating reasoning. Another option would be to consider, e.g., Echo-State Networks (ESN), that have shown promising results in term series classification as well [4].

References

1. Barbieri, D.F., Braga, D., Ceri, S., Della Valle, E., Grossniklaus, M.: Incremental reasoning on streams and rich background knowledge. In: Extended Semantic Web Conference. pp. 1–15. Springer (2010)
2. Barbieri, D.F., Braga, D., Ceri, S., Valle, E.D., Grossniklaus, M.: C-SPARQL: a continuous query language for RDF data streams. International Journal of Semantic Computing **4**(01), 3–25 (2010)
3. Besold, T.R., d'Avila Garcez, A.S., Bader, S., Bowman, H., Domingos, P.M., Hitzler, P., Kühnberger, K., Lamb, L.C., Lowd, D., Lima, P.M.V., de Penning, L., Pinkas, G., Poon, H., Zaverucha, G.: Neural-symbolic learning and reasoning: A survey and interpretation. CoRR **abs/1711.03902** (2017)
4. Bianchi, F.M., Scardapane, S., Løkse, S., Jenssen, R.: Reservoir computing approaches for representation and classification of multivariate time series. IEEE Trans. Neural Networks Learn. Syst. **32**(5), 2169–2179 (2021)
5. Brickey, D., Guha, R. (eds.): RDF Schema 1.1. W3C Recommendation, 23 February (2014)

6. Cho, K., van Merrienboer, B., Gülçehre, Ç., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning phrase representations using RNN encoder-decoder for statistical machine translation. In: EMNLP. pp. 1724–1734. ACL (2014)

7. Cugola, G., Margara, A.: Processing flows of information: From data stream to complex event processing. ACM Computing Surveys (CSUR) **44**(3), 1–62 (2012)

8. Della Valle, E., Ceri, S., Van Harmelen, F., Fensel, D.: It's a streaming world! reasoning upon rapidly changing information. IEEE Intelligent Systems **24**(6), 83–89 (2009)

9. Dell'Aglio, D., Della Valle, E., van Harmelen, F., Bernstein, A.: Stream reasoning: A survey and outlook. Data Science **1**(1-2), 59–83 (2017)

10. Ebrahimi, M., Sarker, M.K., Bianchi, F., Xie, N., Doran, D., Hitzler, P.: Reasoning over RDF knowledge bases using deep learning. CoRR **abs/1811.04132** (2018)

11. Fawaz, H.I., Forestier, G., Weber, J., Idoumghar, L., Muller, P.A.: Deep learning for time series classification: a review. Data Mining and Knowledge Discovery **33**(4), 917–963 (2019)

12. Ferreira, R., Lopes, C., Gonçalves, R., Knorr, M., Krippahl, L., Leite, J.: Deep neural networks for approximating stream reasoning with C-SPARQL. CoRR **abs/2106.08452** (2021)

13. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016)

14. Harris, S., Seaborne, A., Prud'hommeaux, E. (eds.): SPARQL 1.1 Query Language. W3C Recommendation, 21 March (2013)

15. Haykin, S.: Neural networks: a comprehensive foundation. Prentice Hall PTR (1994)

16. Heath, T., Bizer, C.: Linked Data: Evolving the Web into a Global Data Space. Synthesis Lectures on the Semantic Web, Morgan & Claypool Publishers (2011)

17. Hitzler, P., Bianchi, F., Ebrahimi, M., Sarker, M.K.: Neural-symbolic integration and the semantic web. Semantic Web **11**(1), 3–11 (2020)

18. Hitzler, P., van Harmelen, F.: A reasonable semantic web. Semantic Web **1**(1-2), 39–44 (2010)

19. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation **9**(8), 1735–1780 (1997)

20. Hohenecker, P., Lukasiewicz, T.: Ontology reasoning with deep neural networks. J. Artif. Intell. Res. **68**, 503–540 (2020)

21. Makni, B., Hendler, J.A.: Deep learning for noise-tolerant RDFS reasoning. Semantic Web **10**(5), 823–862 (2019)

22. Margara, A., Urbani, J., Van Harmelen, F., Bal, H.: Streaming the web: Reasoning over dynamic data. Journal of Web Semantics **25**, 24–44 (2014)

23. Miller, E.: An introduction to the resource description framework. D Lib Mag. **4**(5) (1998)

24. Motik, B., Grau, B.C., Horrocks, I., Wu, Z., Fokoue, A., Lutz, C. (eds.): OWL 2 Web Ontology Language Profiles. W3C Recommendation, 11 December (2012)

25. Pascanu, R., Mikolov, T., Bengio, Y.: Understanding the exploding gradient problem. CoRR **abs/1211.5063** (2012)

26. Qin, Y., Song, D., Chen, H., Cheng, W., Jiang, G., Cottrell, G.W.: A dual-stage attention-based recurrent neural network for time series prediction. In: IJCAI. pp. 2627–2633. ijcai.org (2017)

27. Ren, X., Khrouf, H., Kazi-Aoul, Z., Chabchoub, Y., Curé, O.: On measuring performances of C-SPARQL and CQELS. In: SR+SWIT@ISWC. CEUR Workshop Proceedings, vol. 1783, pp. 1–12. CEUR-WS.org (2016)

28. Wang, Z., Yan, W., Oates, T.: Time series classification from scratch with deep neural networks: A strong baseline. In: IJCNN. pp. 1578–1585. IEEE (2017)