Abstract—We consider a scenario where an artificial agent is reading a stream of text composed of a set of narrations, and it is informed about the identity of some of the individuals that are mentioned in the text portion that is currently being read. The agent is expected to learn to follow the narrations, thus disambiguating mentions and discovering new individuals. We focus on the case in which individuals are entities and relations and propose an end-to-end trainable memory network that learns to discover and disambiguate them in an online manner, performing one-shot learning and dealing with a small number of sparse supervisions. Our system builds a not-given-in-advance knowledge base, and it improves its skills while reading the unsupervised text. The model deals with abrupt changes in the narration, considering their effects when resolving coreferences. We showcase the strong disambiguation and discovery skills of our model on a corpus of Wikipedia documents and on a newly introduced data set that we make publicly available.

Index Terms—Machine learning, natural language processing (NLP), recurrent neural networks.

I. INTRODUCTION

NOWADAYS, most of machine-learning-based approaches to language-related problems are defined in the classical setting in which we exploit a batch of data to build powerful, offline predictors, especially when the data are paired with supervisions. Little has been done when considering the setting in which we process a continuous stream of text and build systems that learn and respond in an online manner. However, conversational systems [1], information extractors from streams of news [2], or social network data [3], and those systems that require interactions with the environment [4] belong to the latter setting. Supervisions are usually sparse and very limited, and the system is expected to perform “one-shot” learning and to quickly react to them.

This article faces the problem of learning to extract information while reading a text stream, with the aim of identifying entities and relations in the text portion that is currently being read. This problem is commonly tackled by assuming the existence of a knowledge base (KB) of entities and relations, where several entity/relation instances are paired with additional information, such as the common ways of referring to them or sentences/facts in which they are involved. Then, once an input sentence is provided for reading, subportions of text must be linked to entity or relation instances of the KB. The linking process introduces the challenging issue of dealing with multiple distinct entities (relations) that are mentioned with the same text, and thus, the system has to disambiguate which is the “right” entity or relation instance of the KB for the considered text fragment. In particular, the context around the fragment or, if needed, information that was provided in the previous sentences of the text stream can be used to perform the disambiguation. As a very simple example, consider the sentence Clyde went to the office, being Clyde and the office two text fragments that indicate entities, while went to is text that is about a relation. Clyde could be the mention that is used to indicate different people in the KB, and several offices could be mentioned by the expression the office (mentions to relations follow the same logic).

At a first glance, this problem shares basic principles and intuitions with several existing methods, such as entity linking (EL) [5], word sense disambiguation (WSD) [6], named entity recognition (NER) [7], knowledge population [8], and others (we postpone to Section V the description of the related approaches). However, the problem we consider in this article is way more challenging, since the aforementioned KB is not given in advance, and the system has to take care of progressively building it while also using such KB to disambiguate the input data. This means that the system is required to decide whether a certain mention is about an entity/relation instance already inserted into the KB or if it is about a never-seen-before entity/relation and, in that case, to update the KB. Moreover, since we deal with streams of text, learning is performed in an online manner, and the system has to take a decision before processing the following sentence.

Motivated by the aforementioned challenges, in this article, we propose an end-to-end memory-augmented trainable system that learns to discover and disambiguate entity/relation instances in a text stream, exploiting a small number of sparse supervisions and operating in an online manner. In particular, this article makes the following contributions.

1) We propose a new online learning method for populating a not-given-in-advance KB of entities and relations.

Digital Object Identifier 10.1109/TNNLS.2019.2955597
2) We introduce a new scheme to learn latent representations of entities and relations directly from data, either autonomously (self-learning) or using limited supervisions. Character-based encoders are exploited to handle small morphological variations (plurals, suffixes, and so on) and typos, synonymy, and semantic similarity.

3) We present a new problem setting where the system is evaluated while reading a stream of sentences organized into short stories, requiring online learning capabilities.

4) We showcase the validity of our approach both in an existing data set made of Wikipedia articles and a new data set that we introduce and make publicly available.

This article is organized as follows. Section II formally describes the problem we face. The inferential process of the proposed architecture is described in Section III, while online learning dynamics are described in Section IV. Related work is reviewed in Section V. Experiments and future plans are reported in Sections VI and VII, respectively.

II. PROBLEM SETTING

We consider a continuous stream of text at each time step, \( t \) produces a sentence \( s_t \). Groups of contiguous sentences are organized into small stories about a (not-known-in-advance) set of actors/objects so that the narration is discontinuous whenever a new story begins.

We focus on the problem of developing a system that, given \( s_t \), produces its interpretation by linking text fragments to a KB that the system is responsible for creating and updating (see Fig. 1). We think of the KB as a set of instances, and the considered text fragments of \( s_t \) are mentions to them. Some mentions are about entities, and others are about relations. For each instance, the KB stores (possibly) multiple mentions that are commonly used to refer to the instance itself, as for entities 1 and 2 in Fig. 1. On the other way around, the same mention can be shared by more than one instance, as in the example of Fig. 1 where the same textual form Clyde refers to different entities (entities 1 and 6). KB instances also include information about the contexts in which instances have been mentioned in the stream so far, where the notion of context compactly indicates the whole sentence in which the instance was mentioned. Here and throughout this article, we simplify the descriptions by frequently using the generic term instance to refer to both the cases of relations and entities, without making a precise distinction (if not needed).

We consider the case in which text fragments of \( s_t \) are matched with the mentions in the KB to detect compatible instances. Then, instances are disambiguated by observing the current context (that is the portion of \( s_t \) around the mention) and exploiting the knowledge about the story to which \( s_t \) belongs (up to the current sentence). At the beginning of each story, a small number of sentences are supervised with the identity of the mentioned entities and relations. The system is expected to follow the narration by disambiguating mentions, learning from such sparse supervisions (and quickly reacting to them), and discovering new instances. The natural ambiguity of language, the discontinuous narration, and the dynamic nature of the KB require the system to develop advanced disambiguation procedures to interpret \( s_t \).

It is worth noticing that, in the proposed setting, the system decisions on a given sentence are evaluated immediately when processing the following sentences, regardless of whether the sentence was supervised or not. This schema gives rise to a novel online dynamics of the learning process, which is profoundly different from most of the existing approaches, being it more challenging and realistic than common batch-mode offline approaches (consider, for example, the case in which it is framed within conversational applications). This article encourages the development of new methods, models, and data sets for this extremely relevant but largely unexplored setting.

The proposed model has the potentiality of being enhanced by introducing a more structured and advanced knowledge component (types, facts, and abstract notions for higher level inference)—such as logic formulas). For example, once entities and relations are detected from a sentence \( s_t \), they could be combined together to create symbolic representations of facts. Thus, the KB could be represented as a knowledge graph, where entities are vertices and relations are the edges of the graph. As long as the KB becomes more structured, the added information may be used to enhance the disambiguation procedure or to introduce a further level of abstraction on which symbolic reasoning can be performed. In this article, we do not face the problem of designing an enhanced KB module, but we focus on the tasks of instance discovery and disambiguation in text streams, which provides the basic functionalities required to link the input text to the instances in the KB.

Section III describes our model from an operational point of view, while Section IV describes the learning dynamics associated with the online learning framework.

III. MODEL

At each time step, the system processes an input sentence \( s \) (we drop the time index for simplicity), and it detects
those text portions \( z \) (i.e., one or more adjacent tokens) that are expected to mention KB instances. This task is referred to as mention detection. For each candidate mention \( z \) in \( s \), the system predicts how much each KB instance is compatible with \( z \) in the context where it appears in \( s \). The prediction scores are collected in a vector \( \alpha \), where each entry corresponds to a given instance in the KB. Then, the mention is linked to the most-likely instance \( y \) as

\[
\alpha = \phi(z, s), \quad y = \arg \max(\alpha)
\]  

where \( \phi \) is the function that computes the affinity scores of \( z \) with respect to all the KB instances, given the context of the current sentence \( s \). In this article, we replace the second argument of \( \phi \) by the sequence of mentions detected in \( s \), excluding \( z \) itself, i.e., \( \phi(z, \{z_j : z_j \in s, z_j \neq z\}) \).

The whole system is the composition of multiple computational modules that we sketch in Fig. 2. The MENTION DETECTOR segments \( s \) by identifying mentions to entities and relations. For example, Parry is chasing a mouse is segmented into two mentions to entities (i.e., “Parry” and “a mouse”) and a mention to a relation (i.e., “is chasing”). Each mention \( z \) and its context are encoded into vectorial representations (i.e., embeddings) by the ENCODER module. The CANDIDATE GENERATOR produces a probability distribution over the KB instances, by combining different sources of information (e.g., surface form of the mention, embedding, and temporal coherence in the current story). Finally, a DISAMBIGUATOR takes the final decision on which is the most likely KB instance to link, using the aforementioned probability distribution and the embedding of the mention context. Sections III-A–III-D describe each computational block in detail.

### A. Mention Detection

The goal of the MENTION DETECTOR (first block of the pipeline in Fig. 2) is to segment each sentence into nonoverlapping text fragments, which are mentions to yet-unknown entity or relation instances.

Motivated by the need of developing models that are robust to morphological changes and that do not depend on a predefined vocabulary of words (as needed by interactive/conversational applications), we process the input data using a character-level encoding, following the approach we proposed in [9]. In particular, bidirectional recurrent neural networks (BiRNNs) are exploited to build vectorial representations of words at multiple levels. Given an input sentence \( s \), composed of \( v \) words \( x_i, i \in [1, v] \), the word embedding \( e_{i}^{w} \) of the word \( x_i \) is

\[
e_{i}^{w} = \tilde{r}_{e,w}(\hat{c}_{i,1} \ldots \hat{c}_{i,|s|})
\]

where \( \hat{c}_{i,k} \) is the \( k \)th character of the \( i \)th word and \( \tilde{r}_{e,w} \) is a BiRNN outputting the concatenation of its hidden states in both directions. Since the morphological representation of a word is usually meaningless if taken isolated from its context, we compute the contextualized word embedding \( \tilde{e}_{i}^{w} \) of the word \( x_i \) as

\[
\tilde{e}_{i}^{w} = [\tilde{r}_{e,1}^{w}(e_{1}^{w} \ldots e_{i}^{w}), \tilde{r}_{e,\bar{w}}^{w}(e_{\bar{w}}^{w} \ldots e_{i}^{w})]
\]

where \( \tilde{r}_{e,1}^{w} \) and \( \tilde{r}_{e,\bar{w}}^{w} \) are two RNNs processing the character-level embeddings of the words in the left and right contexts of \( x_i \) within the sentence \( s \), including \( x_i \) itself. \( \tilde{r}_{e,1}^{w} \) and \( \tilde{r}_{e,\bar{w}}^{w} \) output their hidden state, and we denote with \([\cdot, \cdot]\) the concatenation operation. All the RNNs used in this article are long short-term memories (LSTMs).

The output of this computational block is based on the predictions of an MLP classifier that processes (one-by-one) the contextualized embeddings of the words in the input sentence, computed as in (3). The MLP is trained using supervised learning, with a tagging scheme similar to [10]. In particular, each word needs to be classified as being the \( \text{begin} \), \( \text{inside} \), \( \text{end} \) word of either an entity or a relation mention, for a total of six classes. As a result, each mention \( z \) is composed by the sequence of words where the first word is tagged with the \( \text{begin} \) tag, the last word with the \( \text{end} \) tag, and the other words are predicted as inner (tags must be all of the same type, either entity or relation).

As remarked in Sections I and II, we exploit sparse supervisions so that training the MLP-tagger (and, in turn, the embedding networks \( r_{c,w} \) and \( r_{e,w} \)) might be difficult. However, we follow the intuition that syntax has a crucial role in text segmentation; noun phrases are mentions to entities, while fragments that start with a verb and end with a preposition (if any) are mentions to relations [11].\(^2\) Hence, we can use these rules to automatically generate artificial supervisions on large collections of text to pretrain the MENTION DETECTOR.

### B. Mention and Context Encoding

Once mentions have been detected, the ENCODER module (second block of Fig. 2) encodes them and their contexts into vectorial representations, following an encoding scheme similar to the one in Section III-A. However, after the processing of the previous computational block, the input sentence becomes a sequence of mentions, each composed of one or more words. In detail, at this stage, \( s \) is a sequence of \( m_s \) mentions \( z_{i}, i = 1, \ldots, m_s \). Two different vectorial representations are computed for each mention. First, the mention embedding \( e_{i}^{m} \) of \( z_{i} \) is obtained as

\[
e_{i}^{m} = \tilde{r}_{c,m}^{w}((\hat{c}_{i,1} \ldots \hat{c}_{i,|s|}))
\]

where \( \hat{c}_{i,1} \ldots \hat{c}_{i,|s|} \) is the sequence of characters of the mention.

\(^2\)See the Supplemental Material for the details.
Second, the context embedding \( \hat{e}_i^m \) is computed from the other mentions \( z_j, j \neq i \) in \( s \), as
\[
\hat{e}_i^m = \left[ r_{e,m}(e_1^m, \ldots, e_{i-1}^m), \frac{r_{e,m}(e_n^m, e^m)}{r_{e,m}(e^m)} \right].
\] (5)
Notice that, differently from (3), here we are encoding only the context around the considered mention, excluding the mention itself.

In order to learn the parameters of the encoders \( r_{e,m} \) and \( r_{e,m} \), we follow the principles at the basis of the Word2Vec-CBOW architecture [12], which is rooted on the idea of encoding the context around a word and decoding the word itself, to generate representations where synonyms or semantically similar words are close to each other in the embedding space. However, differently from the original CBOW, we use character-level input encoding [see (4)] so that we also expect morphologically similar inputs to be close in the embedding space. We also introduce another feature that makes our approach different from other related systems (see [9], [13]), that is, we keep also the decoding stage at a character level, thus avoiding the need of a large mention vocabulary.

In particular, once the context around mention \( z_i \) is encoded into the context embedding \( \hat{e}_i^m \) [see (5)], we exploit an LSTM-based decoder whose initial state is set to \( \hat{e}_i^m \), which generates the sequence of characters that compose \( z_i \). The encoder and the decoder can be trained by exploiting the cross-entropy loss function over character predictions. Once the MENTION DETECTOR has been pretrained accordingly to what suggested in Section III-A, the ENCODER module can be pretrained as well without any human intervention, by processing large collections of text and learning to decode each mention.

C. Candidate Generation

Given an input mention \( z \) from the current sentence and its embedding \( \hat{e}_i^m \), the CANDIDATE GENERATOR (third block in Fig. 2) implements four memory components that are used to generate a list of candidate KB instances compatible with \( z \) and, afterward, to store the information on the disambiguated instance. Before providing further details on the candidate generation process, we describe the four memory components, as shown in Fig. 3.

1) The memory component \( \mathcal{H} \) is an ordered set that collects all the mentions that were processed up to the current sentence; this allows a fast lookup of the previously predicted instances for specific mentions. For example, if the mention John Doe was previously assigned to instance \( k \), when processing the same mention again, the system could easily hypothesize that it still belongs to instance \( k \).

2) The matrix \( E \) stores (row-wise) the embeddings of the mentions in \( \mathcal{H} \), computed with the encoder of Section III-B [see (4)]. Due to a similarity measure in the embedding space, this component allows the system to associate KB instances to never-seen-before mentions that are small variations of previously seen ones or that refer to semantically similar elements. For example, given the never-seen-before mention John D., the system could easily predict it still belongs to instance \( k \), since its char-level embedding is close to the one of John Doe, even though the exact lookup in \( \mathcal{H} \) failed.

3) The set \( T \) keeps track of the last disambiguated instances (with repetitions). This memory naively allows the system to handle coreferences. As we will see shortly, the system can learn that some specific inputs (e.g., pronouns, category identifiers, and so on) are often assigned to recently mentioned instances, making valuable temporal hypotheses when it has to disambiguate such inputs.

4) The matrix \( M \) stores (row-wise) the instance-activation scores of each mention in \( \mathcal{H} \). Each row is associated with a mention in \( \mathcal{H} \), and each column corresponds to a KB instance. The row of \( M \) associated with a certain mention \( u \in \mathcal{H} \) models how strongly each KB instance is associated with the mention \( u \). The matrix \( M \) is learned while reading the text stream, as we will describe in Section IV.

In the following, we will show how the CANDIDATE GENERATOR exploits these memory components to generate candidates of KB instances, given an input mention. For the purpose of this description, we suppose that the current memory consists of \( n \) instances and \( m \) mentions. The candidate generation routine first outputs three hypotheses that are represented by three vectors \( p^{(z)} \), \( p^{(e)} \), and \( p^{(i)} \) of \( n \) scores each. Each score is in \([0,1]\), and it represents how strongly an instance is a candidate for being linked to the mention \( z \). For example, \( p^{(i)}_j \) [the \( i \)th component of \( p^{(i)} \)] models the probability of the instance \( i \) to be a link candidate according to the first hypothesis. Then, the three hypotheses are combined into a final vector \( p \). A visual representation of this combination is shown in Fig. 4.

![Graphical representation of the four memory components](image-url)

Fig. 3. Graphical representation of the four memory components \( \mathcal{H} \), \( E \), \( M \), and \( T \). \( \mathcal{H} \) collects the (lowercase) mentions that are stored in the KB. For each of them, the embedding vector (blue rectangle) is stored in a row of the matrix \( E \). The affinity scores of the considered mention, with respect to each of the \( n \) instances in the KB, are stored in a row of the matrix \( M \). Finally, \( T \) contains the KB instances that have been recently linked by the system (here represented as a histogram of the number of times that each KB instance was recently linked).
The first vector, \( p^{(c)} \), named STRING-MATCH HYPOTHESIS, contains the activation scores of the \( n \) instances, given the string \( z \). It models the idea that the surface form of \( z \) is a strong indicator for spotting candidate links. Formally

\[
p^{(c)} = \sigma(M_h(z, \mathcal{H}))
\]

where the function \( h(z, \mathcal{H}) \) returns the index of \( z \) in \( \mathcal{H} \), \( \sigma(\cdot) \) is a sigmoidal function that operates element-wise on its input [yielding output values in \((0, 1)\)], and the subscript after the matrix \( M \) indicates the matrix row.

The second hypothesis, \( p^{(e)} \), named EMBEDDING-MATCH HYPOTHESIS, collects the activations of the instances, given the embedding \( e^m \) of \( z \). The rationale behind it is that if \( e^m \) is similar to an embedding of a known mention (in the sense of the cosine similarity \( \cos(\cdot, \cdot) \)), then it is likely to activate the same instances. Due to the way that our embeddings are computed (see Section III-B), we expect that two embeddings are “close” if they have similar roles in the processed sentences (semantic similarity and synonymy) and similar morphological properties (due to the character-level input). Formally

\[
p^{(e)} = \left( \frac{\cos(e^m, E_i) + 1}{\sum_{j=1}^{m} \cos(e^m, E_j) + m} \right)_i^m \cdot \sigma(M)
\]

where the notation \([v_i]_i^m\) indicates the (column) vector \([d_1, \ldots, d_m]\). Notice that \( \sigma(M) \) is a matrix of the same size of \( M \) and \( p^{(e)} \) involves a vector-times-matrix operation that basically computes a weighted sum of the rows of \( M \) accordingly to the similarity between \( e^m \) and the stored embeddings.\(^3\)

The third hypothesis, \( p^{(t)} \), named TEMPORAL HYPOTHESIS, implements the idea that recently disambiguated instances are good candidates for coreference resolution (temporal locality). In other words, if a story is talking about a certain entity, it is likely that the narration will make references to it using some new surface forms. For example, given an entity labeled Donald Trump, there could be ambiguous mentions, such as Donald, Mr. Trump (other people called this way), or the president (that cannot be captured by the other two hypotheses). In these cases, the temporal locality has a crucial role, which is even more evident when using pronouns that are shared by several instances. Formally, we have

\[
p^{(t)} = \frac{\sum_{i=1}^{n} u(i, T)}{\max_{j=1}^{n} [u(j, T)]}
\]

where \( u(j, T) \) returns the number of occurrences of instance \( j \) in \( T \).

The CANDIDATE Generator merges the three hypotheses and produces the FINAL HYPOTHESIS \( p \). The idea behind this operation is to give more priority to \( p^{(c)} \) than to \( p^{(e)} \) when the former is strongly activated (since \( p^{(c)} \) is about “exact” matches in terms of surface forms). The importance of the temporal component \( p^{(t)} \) in the merge operation depends on the current mention \( z \). For example, if \( z \) is a pronoun, the system must trust \( p^{(t)} \) more than the others, while, in case of some unambiguous mention, it must learn that \( p^{(t)} \) is not important. We let the system learn the importance of \( p^{(t)} \) depending on the mention embedding \( e^m \). Formally, the system computes \( \gamma = q(e^m) \in [0, 1] \), where \( q(\cdot) \) is a learnable function (whose form will be defined shortly), and the vector of merged hypotheses is

\[
p = (1 - \gamma) \cdot (p^{(c)} + (1 - p^{(c)})p^{(e)}) + \gamma \cdot p^{(t)}
\]

The vector \( p \) contains a set of scores that model how strongly each KB instance is related to the current input mention.

We kept the model as simple as possible by considering only the three hypotheses that were necessary to cover all the ambiguities present in the task at hand. However, any number of hypotheses can be attached to the candidate generation module, making our model adaptable to different natural language processing (NLP) problems.

\[\text{D. Disambiguation}\]

While the candidate generation routine only focusses on the mention \( z \), the DISAMBIGUATOR [see Fig. 2 (fourth block)] is responsible of determining what are the most likely KB instances given the context of \( z \). The representation of the context \( \hat{e}^m \) is computed by the ENCODER (see Section III-B) by (5). The DISAMBIGUATOR is based on the functions \( d_i(\hat{e}^m), i = 1, \ldots, n \), also referred to as disambiguation units.

In detail, each \( d_i \) is associated with a KB instance, and it is learned while reading the text stream in a supervised or unsupervised way, as we will describe in Section IV. In particular, in the considered problem, we do not have the use of any discriminative information; when we receive the supervision

\[^3\text{In our implementation, we kept only the top-}k\text{ cosine similarities} (k = 5), \text{forcing the other ones to} -1\.]
that the input mention is about a certain KB instance (or when the model decides this in an unsupervised manner), we cannot infer that the context of the considered mention is not compatible with any other KB instance. As a matter of fact, the same context may be shared by several instances, so each \( d_i(\cdot) \) must have the capability of learning from positive examples only. For this reason, we implement \( d_i \) with a locally supported similarity measure

\[
d_i(\hat{\mathbf{e}}^m) = \frac{1}{2} + \frac{1}{2} \max_{j=1, \ldots, \kappa} \cos(\hat{\mathbf{e}}^m, \mathbf{w}_{ij}) \in [0, 1]
\]

that models the distribution of the contexts for instance \( i \) by means of \( \kappa \) centroids \( \{\mathbf{w}_{ij}\} \). As we will describe in Section IV, these centroids are developed in an online manner, and in the unsupervised case, we end up in an instance of online spherical K-Means. Also, the previously mentioned function \( q(\cdot) \) needs to be locally supported (for the same reasons), so we implemented it following (9) as well.

We combine the activations of candidates (i.e., the vectors \( \mathbf{p} \) and the disambiguation unit outputs, \( \mathbf{d} = [d_i(\hat{\mathbf{e}})]_{i=1}^\kappa \), to get the output \( \mathbf{o} \) of the system

\[
\mathbf{o} = \delta(\mathbf{p} > \tau_r) \cdot (\eta \cdot \mathbf{p} + (1 - \eta) \cdot \mathbf{d})
\]

where \( \delta \) returns a binary vector with 1s in those positions for which the (element-wise-evaluated) condition in bracket is true. The scalar \( \tau_r > 0 \) is a reject threshold, and \( \eta \in [0, 1] \) is a tunable parameter that controls the role of the hypothesis \( \mathbf{p} \) in the decision process. The reject threshold allows us to avoid computing \( d_i(\cdot) \) associated with very-low-probability candidate instances.

**IV. ONLINE LEARNING DYNAMICS**

The system reads data from a text stream and it optimizes the model parameters in an online manner. Before going into further details, we recall that the learnable parameters of the proposed model are the matrix \( M \) in the memory component, the vectors \( \mathbf{w}_{ij} \) of the disambiguation units and the temporal relevance function \( q(\cdot) \) (i.e., the parameters of the CANDIDATE GENERATOR and of the DISAMBIGUATOR), and the weights of the LSTMs in the MENTION DETECTOR and in the ENCODER.

The system starts with an empty KB (so \( M \) is not allocated yet) and with randomly initialized model parameters. As already introduced in Sections III-A and III-B, we can pre-train those modules that constitute the preliminary stages in the system pipeline of Fig. 2, i.e., the MENTION DETECTOR and then the ENCODER (in both cases, without human intervention). In other words, the system will start reading data from the text stream and it will progressively acquire the skill of detecting mentions to entities and relations, and the skill of encoding such mentions and their context. When a stationary condition is reached in the detector and encoders, the system can start to develop also the KB-based disambiguator and to eventually refine and improve the pretrained modules.

While processing the text stream, accordingly to (1), each detected mention \( z \) is associated with a disambiguated KB instance \( y \). Before starting the disambiguation, the system verifies if \( z \) is already in \( \mathcal{H} \). If it is not the case, then \( z \) is included in \( \mathcal{H} \), its embedding \( \mathbf{e}^m \) is appended to \( E \), and a new row is added to \( M \) (with values such that \( \sigma(M_{h(\cdot), \tau_0}) \approx 0 \)). The learning stage consists in an online process to optimize the model parameters accordingly to either SELF-LEARNING or a SUPERVISION about the target instance \( \bar{y} \). A sketch of the whole learning stage is shown in Algorithm 1.

**Algorithm 1 Learning Dynamics**

```plaintext
1  foreach mention in the current sentence do
2      if supervision not provided then
3          if recognized some instances then
4              reinforce the associated disambiguation units
5          end
6          else if uncertain disambiguation then
7              no actions
8          end
9          else if no recognized instances then
10             create new instance
11             reinforce the new disambiguation unit until > \( \tau_d \)
12          end
13      end
14      else
15          if already known supervision then
16              reinforce the associated disambiguation unit until > \( \tau_a \)
17              penalize the other disambiguation units until < \( \tau_a \)
18          else
19              disambiguate instance \( y \)
20              if \( y \) was associated to another supervision then
21                  goto line 10
22              end
23              else
24                  associate supervision to the \( y \)-th disambiguation unit
25                  goto line 16
26              end
27          end
28      end
29  end
```

A. Self-Learning

When no supervision is provided, learning dynamics changes in function on the confidence that the system yields in formulating hypotheses (\( \mathbf{p} \)) and in disambiguating the mention \( z \) (\( \mathbf{o} \)). We distinguish among three cases as follows.

1. \( \max \mathbf{o} \geq \tau_0 \): RECOGNIZED SOME INSTANCES.
2. \( \max \mathbf{p} > \tau_r \wedge \max \mathbf{o} < \tau_a \): UNCERTAINTY.
3. \( \max \mathbf{p} \leq \tau_r \): UNKNOWN INSTANCE

where \( \tau_r \) is the aforementioned reject threshold and \( \tau_a \in (\tau_r, 1) \) is an accept threshold. In case 1), the response of

\[4\text{For simplicity, we set } \eta = 0.5.\]

\[5\text{We set } \tau_r = 0.1 \text{ and } \tau_a = 0.9.\]
the system has been rather strong in indicating at least one instance, and therefore, the prediction of the model is considered reliable enough, so such decision must be reinforced for all those disambiguation units that generated outputs in $o^*$ above the accept threshold $\tau_a$. This is done in a self-learning fashion [14], by means of a single online gradient-based update, with the aim of minimizing the quadratic loss that measures the distance between the selected disambiguation unit outputs (indexed by $j$) and 1, that is, $\sum_j (d_j - 1)^2$ (i.e., we reinforce the selected outputs). In case 2), the system activates some candidates, but it is uncertain in the disambiguation, and thus, no further actions are taken. Case 3) is triggered when $p$ is composed of only low-confidence candidate activations. This situation happens when the candidate generation module does not find a known instance that is compatible with the current mention, which is likely to indicate the occurrence of a new entity/relation. Therefore, the system creates a new instance in the KB and reinforces its disambiguation unit until its response is above $\tau_a$ to develop the new instance model (i.e., multiple gradient-based updates).

B. Supervision

When a supervision $\mathbf{y} \in \mathcal{Y}$ is provided for the mention $z$, we want the system to immediately learn from it. The system keeps track of the mapping between the set of user-provided supervisions $\mathcal{Y}$ and the set of instances in the memory components (the user is not aware of how the system handles instances). When $\mathbf{y}$ is a known supervision, the index of the corresponding instance is found, and the output of the disambiguation unit associated with this instance is reinforced until it is greater than the accept threshold $\tau_a$. We also push the output of the other disambiguation units toward 0, by minimizing the quadratic loss, $\sum_j (d_j)^2$. This implements a penalization process, which involves multiple gradient-based updates until all the involved outputs fall below $\tau_a$. When $\mathbf{y}$ is a never-seen-before supervision for the system, then it is associated with the disambiguated instance $y$. On the other hand, if $y$ was previously associated with another supervision symbol in $\mathcal{Y}$ different from $\mathbf{y}$, then we have a collision in the mapping and solve it by creating a new instance and by associating it to $\mathbf{y}$. Then, we follow the same steps as in case 3).

V. Related Work

Mining over text streams has been studied in a number of works [15]–[17], with several different purposes, which, however, are different from what we consider in this article. Our approach to the learning problem is based on simple sentences that have the same structure of the ones used in many tasks of the bAbI project by Facebook [18], [19]. However, none of such tasks is conceived for online learning or for entity/relation extraction and disambiguation. Interesting ideas on entity-oriented subsymbolic memory components have been recently proposed in [20] and [21] and extended to the case of relations in [22]; their formulation is developed to comply with the aforementioned bAbI asks. The idea of considering small text passages could resemble the task of machine comprehension, where, however, such passages are read with the purpose of answering a question [23]–[25]. Concerning the input representation, we took inspiration from those works that exploit character-level embeddings to build models that also consider morphological information [13], [26].

Our approach disambiguates mentions using their contexts, so it shares several aspects with WSD and EL, which, differently from our case, assume to work with a given KB. In WSD [6], [27], the set of target words is known and the senses of each word are taken from a given dictionary. EL [5] is similar to WSD [28], [29], but it is about linking "potentially partial" entity mentions to a target KB that has an encyclopedic nature [29], [30]. The EL problem is presented in several variants and focusing on different types of data [31]–[36], and it has been the subject of task-oriented evaluation procedures and benchmarks [37], [38]. A few EL systems work in an unsupervised way [33], [39], but the KB is still given. NER focuses on discovering mentions to entities, and it is also a basic module of several EL systems [40]. However, NER is about proper nouns, as frequently is EL [31], while here we also consider common nouns. Moreover, NER systems output the entity type (person, location, and so on) without producing any instance-level information [7], [10]. Relation extraction (RE) has been recently approached with end-to-end and advanced embedding-based models [41], [42]. The entities involved in the target relation are usually known, and a predefined ontology is given (distant supervisions are also used, as in [43]). There are a number of discussions to better state the RE problem and build accurate gold labels [44].

Finally, learning the KB component is the subject of those tasks of automatic KB construction [45] and KB population [46], [47], which, differently from the case of this article, either makes some application-specific assumptions to implement the KB or exploit a given ontology schema, also combining unsupervised and supervised learning with ensembles and stacking techniques [8].

VI. Experiments

A. Data Sets

1) Simple Story Data Set: A detailed experimentation was carried out in a new data set that we created and made publicly available at http://sailab.diism.unisi.it/stream-of-stories/. We remark that the problem we face shares some aspects with the existing benchmarks, but none of them is really focusing on what we introduced in Section II. The data we created are composed of a stream of 10 000 sentences that organized into 564 stories (similar to what reported in Fig. 1). Each story is composed of a list of not-repeated facts, involving 130 entity
Fig. 5. Ontology graph of entity (nodes) and relation (edges) types in the Simple Story data set. Dashed lines between node pairs indicate a hierarchy between the two types.

TABLE I
ACCURACY (%) FOR DIFFERENT AMOUNTS OF SUPERVISION IN THE WIKIFACTS DATA SET

| Model       | 10%   | 25%   | 50%   | 90%   |
|-------------|-------|-------|-------|-------|
| All         |       |       |       |       |
| RB          | 16.84 | 40.44 | 48.28 | 49.55 |
| Deep-RNN    | 0.6   | 3.01  | 12.34 | 21.78 |
| Our Model   | 39.25 | 54.57 | 69.64 | 75.45 |
| Re-Reading  | 44.75 | 54.66 | 66.88 | 70.55 |
| Last        |       |       |       |       |
| RB          | 17.28 | 40.87 | 48.04 | 49.37 |
| Deep-RNN    | 0.6   | 3.25  | 12.11 | 21.37 |
| Our Model   | 37.44 | 52.93 | 67.45 | 75.37 |
| Re-Reading  | 43.41 | 53.38 | 65.13 | 70.39 |

and 27 relation instances that belong to a predesigned ontology (not provided to the system) shown in Fig. 5. Facts in a story mostly talk about a certain entity, which we refer to as “main entity” and can also appear in other stories. Entities and relations are mentioned with different surface forms (synonyms, subportions of names, and so on).

In particular, a sentence is constituted by a triple of mentions, involving two entities and a relation. We automatically generated the data, after having defined the aforementioned ontology with 21 and 28 entity and relation types (see Fig. 5). Different kinds of noise are introduced, from character-level perturbations (simulating typos) to nonmain-entity-related story facts (to make descriptions slightly depart from the main subject). The resulting data set consists of 2176 unique single word tokens, and dictionaries of 1526 and 288 mentions to entities and relations, respectively, which include different variations (typos, determiners, and so on) of 354 base entity mentions and 176 base relation mentions. There are 7975 co-references (including pronouns). Finally, 6830 mention occurrences are ambiguous (i.e., refer to multiple instances).

2) WikiFacts: WikiFacts is a data set proposed in [48] where Wikipedia pages are loosely aligned with Freebase triples. It is composed of a collection of summaries, each of them being the textual description of a certain entity, belonging to the domain of movies. The textual span where an entity is mentioned on Wikipedia summaries is annotated with its identifier on Freebase, and the Freebases fact to which it participates. Relations are not explicitly segmented in the text. Overall the data set contains about 560k entities extracted from 10k pages. WikiFacts can be adapted to the problem setting that we consider in this article (see Section II). In particular, we focused on a subportion of data containing 10k Freebase facts (1112 pages) and used each summary as a story, keeping only stories longer than three sentences. We considered all the sentences that include at least two entities and artificially marked as relation the text span between two consecutive entities. Since relations are not annotated with any Freebase identifiers, we only measure the accuracy on linking mentions to entity instances. A remarkable difference between the previous data set and WikiFact is that, in the latter, entities are less often repeated among the stories. There are 4431 entities in 10k facts, with respect to the 130 entities in the previous data collection.

B. Learning Settings

We split each story into two parts: a supervised and an unsupervised one. The supervised portion covers the first sentence of the story, and in different experimental settings, it consists of 10%, 25%, 50%, and 90% of the story sentences. The system reads the data, receives supervisions, and makes predictions on the unsupervised sentences, according to the stream order. The accuracy on each prediction is measured at the same time when the prediction is made. The results are reported by considering all the unsupervised sentences of a story (ALL—this set of sentences is different in function of the supervised part size) and only the last sentence of the story (LAST—this set is the same for all the supervised part sizes). They are averaged over each story first and, then, over the whole set of stories. We use the same criterion of the cluster purity measure [49] to map unsupervised outputs to the ground truths, where, in case of conflicting assignments, we only keep the mappings that are determined using the largest statistics. The map used to convert predictions is computed with the statistics accumulated up to the time when the prediction is made.

Our model\(^8\) was bootstrapped accordingly to the scheme mentioned in Section IV. In particular, before streaming the stories, we generated a stream of text composed of simple language sentences, taken from Simple English Wikipedia and the Children’s Book Test (CBT) data,\(^9\) getting overall a total of 1.5 million sentences. We automatically generated supervisions for the mention detector according to the procedure described in Section III-A and processed the stream. Afterward, we stopped updating the mention detector and streamed the same data.

\(^8\)The details of the architecture that we selected are reported in the Supplementary Material.

\(^9\)https://simple.wikipedia.org, https://research.fb.com/downloads/babi/
Fig. 6. Accuracy (%) for different amounts of supervision in the Simple Story Data set, in the case of entities (two leftmost graphs) and relations. We include two competitors (Deep-RNN and RB), our system, and our system reading the stream again.

again to allow the system to develop the mention and context encoders (see Section III-B). In both the cases of the detector and the encoders, we randomly injected a char-level noise (typos) to make the models more robust to these kinds of perturbations. Finally, we stopped updating the encoder and started to stream the stories.

C. Competitors

We compared our model with two competitors. The first one, Deep-RNN, is a deep neural architecture, where bottom layers completely coincide with our ENCODER module, and thus, they are two stacked BiRNNs operating on characters and mentions levels, respectively (they are pretrained as in our model). In order to classify mentions, the Deep-RNN architecture uses an MLP (one hidden layer with 600 units and tanh activation; softmax activation in the output layer) on top of the concatenated representation $[e^m, \hat{e}^m]$ [see (4) and (5)]. This network knows in advance the number of instances of the data sets, that is, the size of the output layer of the MLP, and it does not incur in errors related to the self-discovering of new instances. We followed a classic online supervised learning scheme, where a single gradient-based update is performed for each processed sentence; otherwise, we experimented that the network simply overfits the last supervisions and forgets about the rest. We also considered a very informed rule-based model, RB, which buffers statistics on the supervisions received up to time $t$. Given an input mention, RB predicts the most-common supervision for it. When never-supervised-before mentions are encountered, RB predicts the most-frequent supervision of the story, which is likely to be the main entity of the story itself. This information is very precious (our model has no access to it) since several coreferences refer to the main entity. We tested a large number of different rule-based classifiers, and RB was the one leading to the best results. Finally, we report another variant of our model, i.e., the case in which the system reads the whole stream another time (“Re-Reading” without providing supervisions again), that is, the setting in which the self-learning skills are emphasized.

D. Results

Fig. 6 shows the accuracy of discovering and disambiguating mentions to entity and relation instances in the Simple Story data set, while Table I shows the accuracy in the case of the WikiFacts data set (entities). Our system outperforms the competitors in all our experiments. Differently from Deep-RNN, our model exploits its capability of building local models of the instances, while Deep-RNN is not able to capture this locality, either not-learning enough or overfitting supervisions. Moreover, Deep-RNN has difficulties in storing information on the whole story, as shown in the LAST case.

In the case of entities with few supervisions, RB shows an accuracy that is similar to the one of our system on the Simple Story data set. Differently, the proposed model significantly outperforms RB on WikiFacts, showing interesting generalization capabilities on real-world data. In the case of relations, RB is not different from our system, mostly due to the fact that relations are not so ambiguous in the Simple Story data set, also confirmed by the improved results of the Deep-RNN. In most of the cases, the classification capabilities of our model improve when it is allowed to read the stream a second time (rereading), comparing favorably with all the other approaches. This property is more evident in the few supervision settings. These results show that, despite the dynamic nature of the online problem we face, the proposed model has strong memorization skills and the capability of improving its confidence by self-learning.

In the Simple Story data set, we also investigate the behavior of the model at different time instants using 50% of the supervisions (entities). Basically, we paused the system at a certain point of the stream, measured the accuracy, and activated the system again. This process is repeated until the end of the stream is reached. Results are reported in Fig. 7. Our model reaches better results than RB after having read roughly the 20%–25% of the input stream. We analyzed this result, and it turned out that this is mostly due to the fact that our system takes some time to learn how to handle the temporal information ($p^{(t)}$) needed to resolve coreferences. As a matter of fact, coreference resolution clearly depends on the number of supervisions, as reported in Table II.

We also evaluated the values of $\gamma$ that weighs the importance of the temporal locality in the disambiguation process.
In the case of pronouns, the average value of $\gamma$ is 0.33, while in the case of the other mentions (that might include some other coreferences different from pronouns), the average value of $\gamma$ is 0.21, thus showing that the system learns to give more importance to the temporal component when dealing with pronouns than other mentions. This is confirmed by the small instance-activation scores in the case of mentions that are pronouns (the average of max $p^{(z)}$ is 0.005, remarking their inherent ambiguity), with respect to the ones of the other mentions (average of max $p^{(z)}$ is 0.5).

### E. Ablation Study

We compare different variants of the model in order to emphasize the role of each component and report the results in Table III—Simple Story data set. A first comparison regards the benefits of using a character-level encoding for mentions with respect to classical word-level embedding approaches. To this end, we built a vocabulary of words, including all the correct-spelled words of our data set (and an out-of-vocabulary token) and limited the character-based encoder to such words, thus simulating a word-level encoder. Our model is able to encode in a meaningful way those words that contain typos and to exploit them in context encoding, while the word-level encoder faces several out-of-vocabulary words, which also creates ambiguity while comparing contexts.

Another variant of our model discards the temporal hypothesis $p^{(t)}$ when disambiguating entities and considers the hypotheses $p^{(e)}$ and $p^{(e)}$ only. Table III shows that the temporal locality has an important role, and disabling it degrades the performances. This is not only due to its positive effects in coreference resolution but also when disambiguating mentions to the main entity of the story. Finally, due to $p^{(t)}$, the system learns to develop the tendency of associating new mentions to already existing instances, instead of creating new ones, which is an inherent feature of each story (in the worst case, it creates only 183 entity instances).

### F. Dealing With Long Text Streams

The proposed model explicitly memorizes the information about entities and relations of the current and past stories into distinct (i.e., independent) memory locations (see Section III-C). This strongly alleviates the catastrophic forgetting problem [50], which is usually due to memory interference among multiple entities or relations (e.g., due to weight sharing). The experimental analysis given in Section VI-D, where the same stream is read a second time (“Re-Reading”), suggests that the proposed system does not incur into serious catastrophic forgetting issues, since its recognition accuracy improves once reading again the text stream, without additional supervisions (see Fig. 6). Moreover, even if the encoding module in Section III-B is based on RNNs, we are not exposed to the problem of learning long-term dependences with such RNNs [51]. As a matter of fact, the RNNs are only responsible for encoding each mention and its context within a single sentence, whose usual length is efficiently manageable with LSTMs without any long-term dependence problems.

We performed a further experimental analysis that aimed at evaluating the memorization skills of the system when dealing with long text streams. In particular, we provided the system an incremental number of stories and periodically evaluated its capability of correctly recognizing the entities of the first story of the stream. As long as the system reads more data, this procedure provides a clear indication of the persistence of the system memory. However, the system is still exposed to ambiguity issues, since multiple instances might share the same mention. For this reason, we considered different experimental settings with different levels of ambiguity.

In detail, we considered four scenarios in the Simple Story data set, referred to as $A$, $B$, $C$, and $D$, respectively. In each of them, we generated ten different long streams ($\approx 2600$ sentences in each stream), randomly shuffling the involved stories. For each stream, we selected the first story as target story. The model reads the stream and learns in an online manner, receiving supervisions about the mentioned entities and constantly adapting its internal parameters as in the experiments mentioned in Section VI-B. After having processed $t$ stories, we temporarily freeze the whole system and process the target story again, measuring the prediction accuracy on the entities of this story. By increasing the number $t$ of intermediate stories, we evaluate how the systems keep memory of the target story as long as the stream grows.

The four settings differ in how the stream has been constructed, and they introduce an increasing level of ambiguity. In the first setting ($A$), intermediate stories in the stream neither contain entities nor mentions belonging to the target story. Reading the stream does not introduce any ambiguity in the contents of the target story. This allows us to evaluate whether the system is forgetting information as long as time passes, completely discarding ambiguity issues. In setting $B$, we include pronouns in the intermediate stories. Mentions related to pronouns are obviously very ambiguous, and hence, the model will have to rely on the temporal hypothesis (see Section III-C) to be able to disambiguate them. The intermediate stories of setting $C$ can contain mentions that are also used in the target story, and these mentions may or may not refer to the same entities of the target story. In this case, the system must not only remember what was read in the target story but it must also gain stronger skills in disambiguating entities. Finally, in the last scenario ($D$), we ensure that the
intermediate stories never refer to the entities of the target story. However, we still allow the stream to use mentions that are shared with the ones that are used in the target story, simulating a strong data drift.

We explicitly computed the degree of ambiguity of each stream for all the settings (referred to as Amb, Amb, Amb, and Amb), which is the average number of entities associated with each mention of the target story up to step t. Table IV reports both the degrees of ambiguity and the recognition accuracy of the entities of the target story (averaged over the ten generated streams). We report the results after t = 0, 10, 50, 100, and 150 intermediate stories (0, 180, 880, 1700, and 2600 intermediate sentences, respectively). All the four settings show an accuracy close to 100% at t = 0, as expected, where the few errors are mostly due to pronouns, since the system has not developed strong coreference skills, as expected. For t > 0, results in setting A confirm that the model does not forget past information when processing text streams that do not interfere with it. Setting B shows an interesting behavior, in which accuracy is reduced as long as t grows up to 50, and then, it increases again. This indicates that while at the beginning, the system misclassifies pronouns in the target story, it is able to improve pronoun resolution while at the beginning, the system misclassifies pronouns in the target story, it is able to improve pronoun resolution while growing up to 50, and then, it increases again. This indicates that the degradation of the performances is directly related to the degree of ambiguity in the streams. This is a clear hint that long streams increase the difficulty of the task not due to their length but due to their intrinsic ambiguity.

### Table IV

| Setting | Amb | Amb | Amb | Amb |
|---------|-----|-----|-----|-----|
| A       | 1.03 | 1.03 | 1.03 | 1.03 |
| B       | 1.05 | 1.42 | 2.14 | 2.42 |
| C       | 1.04 | 1.50 | 2.49 | 3.28 |
| D       | 1.03 | 1.83 | 3.48 | 4.54 |

VII. CONCLUSION AND FUTURE WORK

We presented an end-to-end model to process text streams, where mentions to entities and relations are detected, disambiguated, and eventually added to an internal KB, which, differently from many existing works, is not-given-in-advance. Our model is capable of performing one-shot learning, self-learning, and it learns to resolve coreferences. It has shown strong disambiguation and discovery skills when tested on a stream of sentences organized into small stories (we also created a new data set that we publicly made available for further studies), even when a few, sparse supervisions are provided. We also showed how it can improve its skills by continuously reading the text. Our future work will focus on exploiting entities and relations structured into facts, higher level reasoning, types, and dynamic reorganization of the KB.

### REFERENCES

[1] Z. Yu, Z. Xu, A. W. Black, and A. Rudnicky, “Strategy and policy learning for non-task-oriented conversational systems,” in Proc. 17th Annu. Meeting Special Interest Group Discourse Dialogue, 2016, pp. 404–412.

[2] G. M. Del Corso, A. Gulli, and F. Romani, “Ranking a stream of news,” in Proc. 14th Int. Conf. World Wide Web, May 2005, pp. 97–106.

[3] A. Ritter, O. Etzioni, and S. Clark, “Open domain event extraction from Twitter,” in Proc. 18th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, 2012, pp. 1104–1112.

[4] K. Christakopoulou, F. Radlinski, and K. Hofmann, “Towards conversational recommender systems,” in Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, 2016, pp. 815–824.

[5] W. Shen, J. Wang, and J. Han, “Entity linking with a knowledge base: Issues, techniques, and solutions,” IEEE Trans. Knowl. Data Eng., vol. 27, no. 2, pp. 443–460, Feb. 2015.

[6] A. Raganato, J. Camacho-Collados, and R.Navigli, “Word sense disambiguation: A unified evaluation framework and empirical comparison,” in Proc. 15th Conf. Eur. Chapter Assoc. Comput. Linguistics, 2017, pp. 99–110.

[7] J. P. C. Chiu and E. Nichols, “Named entity recognition with bidirectional LSTM-CNNs,” Trans. Assoc. Comput. Linguistics, vol. 4, pp. 357–370, Dec. 2016.

[8] N. F. Rajani and R. Mooney, “Combining supervised and unsupervised ensembles for knowledge base population,” in Proc. Conf. Empirical Methods Natural Lang. Process., 2016, pp. 1943–1948.

[9] G. Marra, A. Zugarini, S. Melacci, and M. Maggini, “An unsupervised character-aware neural approach to word and context representation learning,” in Artificial Neural Networks and Machine Learning. Cham, Switzerland: Springer, 2018, pp. 126–136.

[10] G. Lample, M. Ballesteros, S. Subramanian, K. Kawakami, and C. Dyer, “Neural architectures for named entity recognition,” in Proc. NAACL-HLT, 2016, pp. 260–270.

[11] A. Fader, S. Soderland, and O. Etzioni, “Identifying relations for open information extraction,” in Proc. Conf. Empirical Methods Natural Lang. Process., 2011, pp. 1535–1545.

[12] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient estimation of word representations in vector space,” 2013, arXiv:1301.3781. [Online]. Available: https://arxiv.org/abs/1301.3781

[13] R. Jozefowicz, O. Vinyals, M. Schuster, N. Shazeer, and Y. Wu, “Exploring the limits of language modeling,” 2016, arXiv:1602.02410. [Online]. Available: https://arxiv.org/abs/1602.02410

[14] K. Nigam and R. Gholi, “Analyzing the effectiveness and applicability of co-training,” in Proc. 9th Int. Conf. Inf. Knowl. Manage., 2000, pp. 86–93.

[15] A. Banerjee and S. Basu, “Topic models over text streams: A study of batch and online unsupervised learning,” in Proc. SIAM Int. Conf. Data Mining, 2007, pp. 431–436.

[16] B. Krawczyk, L. M. Minku, J. Gama, J. Stefanowski, and M. Woźniak, “Ensemble learning for data stream analysis: A survey,” Inf. Fusion, vol. 37, pp. 132–156, Sep. 2017.

[17] C. C. Aggarwal and C. Zhai, Mining Text Data.

[18] A. Kumar et al., “Ask me anything: Dynamic memory networks for natural language processing,” in Proc. The 33rd Int. Conf. Mach. Learn. Proceedings of Machine Learning Research, M. F. Balcan and K. Q. Weinberger, Eds., vol. 48. New York, New York, USA: PMLR, Jun. 2016, pp. 1378–1387.

[19] M. Henaff, I. Weston, A. Szlam, A. Bordes, and Y. LeCun, “Tracking the world state with recurrent entity networks,” in Proc. ICLR, 2017, pp. 1–14.
X. Ma, N. Fauceglia, Y.-C. Lin, and E. Hovy, “CMU system for entity disambiguation for knowledge base population,” in Proc. 23rd Int. Conf. Comput. Linguistics, 2010, pp. 277–285.

H. Ji and R. Grishman, “Knowledge base population: Successful approaches and challenges,” in Proc. 49th Ann. Meeting Assoc. Comput. Linguistics: Hum. Lang. Technol., vol. 1, Jun. 2011, pp. 1148–1158.

S. Ahn, H. Choi, T. Paranjaa, and Y. Bengio, “A neural knowledge-base model for relation extraction,” arXiv:1608.00318 [Online]. Available: https://arxiv.org/abs/1608.00318

C. D. Manning et al., Introduction to Information Retrieval, vol. 1. Cambridge, U.K.: Cambridge Univ. Press, 2008.

M. McCloskey and N. J. Cohen, “Catastrophic interference in connectionist networks: The sequential learning problem,” Psychol. Learn. Motivat., vol. 24, pp. 105–165, Dec. 1990.

Y. Bengio, P. Simard, and P. Frasconi, “Learning long-term dependencies with gradient descent is difficult,” IEEE Trans. Neural Netw., vol. 5, no. 2, pp. 157–166, Mar. 1994.

Marco Maggini received the M.S. degree (cum laude) in electronic engineering and the Ph.D. degree in computer science and control systems from the University of Florence, Florence, Italy, in February 1991 and 1995, respectively.

He is currently a Full Professor of computer engineering with the Department of Information Engineering and Mathematical Sciences, University of Siena, Siena, Italy. His main research interests are on machine learning, with a focus on neural networks and kernel methods, and applications in information retrieval, information extraction, natural language processing (NLP), and pattern recognition. He has coauthored more than 100 scientific publications.

Stefano Melacci (M’17) received the M.S. degree (cum laude) in computer engineering and the Ph.D. degree in computer science (information engineering) from the University of Siena, Siena, Italy, in 2006 and 2010, respectively.

He worked as a Visiting Scientist at the Computer Science and Engineering Department, The Ohio State University, Columbus, OH, USA. He is currently an Assistant Professor with the Department of Information Engineering and Mathematics, University of Siena. His research interests include machine learning and pattern recognition, mainly focused on neural networks and kernel machines, with applications to computer vision and natural language processing (NLP).

Dr. Melacci serves as an Associate Editor for the IEEE Transactions on Neural Networks and Learning Systems.

Andrea Zugarini received the M.S. degree (cum laude) in computer and automation engineering from the University of Siena, Siena, Italy, in 2017. He is currently pursuing the Ph.D. degree with the Smart Computing Program, University of Florence, Florence, Italy, and University of Siena.

His research interests include machine-learning applications to natural language processing (NLP) with an emphasis on conversational agents.