Published in final edited form as:
Trans Assoc Comput Linguist. 2018 ; 6: 343–356. doi:10.1162/tacl_a_00025.

From Characters to Time Intervals: New Paradigms for Evaluation and Neural Parsing of Time Normalizations

Egoitz Laparra[*], **Dongfang Xu**[*], **Steven Bethard**
School of Information, University of Arizona, Tucson, AZ

Abstract

This paper presents the first model for time normalization trained on the SCATE corpus. In the SCATE schema, time expressions are annotated as a semantic composition of time entities. This novel schema favors machine learning approaches, as it can be viewed as a semantic parsing task. In this work, we propose a character level multi-output neural network that outperforms previous state-of-the-art built on the TimeML schema. To compare predictions of systems that follow both SCATE and TimeML, we present a new scoring metric for time intervals. We also apply this new metric to carry out a comparative analysis of the annotations of both schemes in the same corpus.

1 Introduction

Time normalization is the task of translating natural language expressions of time to computer-readable forms. For example, the expression *three days ago* could be normalized to the formal representation 2017-08-28 in the ISO-8601 standard. As time normalization allows entities and events to be placed along a timeline, it is a crucial step for many information extraction tasks. Since the first shared tasks on time normalization (Verhagen et al., 2007), interest in the problem and the variety of applications have been growing. For example, Lin et al. (2015) use normalized timestamps from electronic medical records to contribute to patient monitoring and detect potential causes of disease. Vossen et al. (2016) identify multilingual occurrences of the same events in the news by, among other steps, normalizing time-expressions in different languages with the same ISO standard. Fischer and Strötgen (2015) extract and normalize time-expressions from a large corpus of German fiction as the starting point of a deep study on trends and patterns of the use of dates in literature.

A key consideration for time normalization systems is what formal representation the time expressions should be normalized to. The most popular scheme for annotating normalized time expressions is ISO-TimeML (Pustejovsky et al., 2003a; Pustejovsky et al., 2010), but it is unable to represent several important types of time expressions (e.g., a bounded set of intervals, like *Saturdays since March 6)* and it is not easily amenable to machine learning (the rule-based HeidelTime (Strötgen et al., 2013) still yields state-of-the-art performance). Bethard and Parker (2016) proposed an alternate scheme, Semantically Compositional

laparra@email.arizona.edu.
[*]These two authors contributed equally.

Annotation of Time Expressions (SCATE), in which times are annotated as compositional time entities (Figure 1), and suggested that this should be more amenable to machine learning. However, while they constructed an annotated corpus, they did not train any automatic models on it.

We present the first machine-learning models trained on the SCATE corpus of time normalizations. We make several contributions in the process:

- We introduce a new evaluation metric for time normalization that can compare normalized times from different annotation schemes by measuring overlap of intervals on the timeline.
- We use the new metric to compare SCATE and TimeML annotations on the same corpus, and confirm that SCATE covers a wider variety of time expressions.
- We develop a recurrent neural network for learning SCATE-style time normalization, and show that our model outperforms the state-of-the-art HeidelTime (Strötgen et al., 2013).
- We show that our character-based multi-output neural network architecture outperforms both word-based and single-output models.

2 Background

ISO-TimeML (Pustejovsky et al., 2003a; Pustejovsky et al., 2010) is the most popular scheme for annotating time expressions. It annotates time expressions as phrases, and assigns an ISO 8601 normalization (e.g., 1990-08-15T13:37 or PT24H) as the *VALUE* attribute of the normalized form. ISO-TimeML is used in several corpora, including the TimeBank (Pustejovsky et al., 2003b), WikiWars (Mazur and Dale, 2010), TimeN (Llorens et al., 2012), and the TempEval shared tasks (Verhagen et al., 2007; Verhagen et al., 2010; UzZaman et al., 2013).

However, the ISO-TimeML schema has a few drawbacks. First, times that align to more than a single calendar unit (day, week, month, etc.), such as *Saturdays since March 6* (where multiple Saturdays are involved), cannot be described in the ISO 8601 format since they do not correspond to any prefix of YYYY-MM-DDTHH:MM:SS. Second, each time expression receives a single *VALUE*, regardless of the word span, the compositional semantics of the expression are not represented. For example, in the expressions *since last week* and *since March* 6, the semantics of *since* are identical – find the interval between the anchor time (*last week* or *March 6*) and now. But ISO-TimeML would have to annotate these two phrases independently, with no way to indicate the shared portion of their semantics. These drawbacks of ISO-TimeML, especially the lack of compositionality, make the development of machine learning models difficult. Thus, most prior work has taken a rule-based approach, looking up each token of a time expression in a normalization lexicon and then mapping this sequence of lexical entries to the normalized form (Strötgen and Gertz, 2013; Bethard, 2013; Lee et al., 2014; Strötgen and Gertz, 2015).

As an alternative to TimeML, and inspired by previous works, Schilder (2004) and Han and Lavie (2004), Bethard and Parker (2016) proposed Semantically Compositional Annotation of Time Expressions (SCATE). In the SCATE schema, each time expression is annotated in terms of compositional time entity over intervals on the timeline. An example is shown in Figure 1, with every annotation corresponding to a formally defined time entity. For instance, the annotation on top of *since* corresponds to a Between operator that identifies an interval starting at the most recent *March 6* and ending at the document creation time (DCT). The Between operator is formally defined as:

$$\text{Between}([t_1, t_2)\colon \text{Interval}, [t_3, t_4)\colon \text{Interval})\colon \text{Interval} = [t_2, t_3)\,.$$

The SCATE schema can represent a wide variety of time expressions, and provides a formal definition of the semantics of each annotation. Unlike TimeML, SCATE uses a graph structure to capture compositional semantics and can represent time expressions that are not expressed with contiguous phrases. The schema also has the advantage that it can be viewed as a semantic parsing task and, consequently, is more suitable for machine-learning approaches. However, Bethard and Parker (2016) present only a corpus; they do not present any models for semantic parsing.

3 An interval-based evaluation metric for normalized times

Before attempting to construct machine-learned models from the SCATE corpus, we were interested in evaluating Bethard and Parker (2016)'s claim that the SCATE schema is able to represent a wider variety of time expressions than TimeML. To do so, we propose a new evaluation metric to compare time normalizations annotated in both the ISO 8601 format of TimeML and the time entity format of SCATE. This new evaluation interprets normalized annotations as intervals along the timeline and measures the overlap of the intervals.

TimeML TIMEX3 (time expression) annotations are converted to intervals following ISO 8601 semantics of their value attribute. So, for example, 1989-03-05 is converted to the interval [1989-03-05T00:00:00, 1989-03-06T00:00:00), that is, the 24-hour period starting at the first second of the day on 1989-03-05 and ending just before the first second of the day on 1989-03-06. SCATE annotations are converted to intervals following the formal semantics of each entity, using the library provided by Bethard and Parker (2016). So, for example, Next(Year(1985), SimplePeriod(YEARS, 3)), the 3 years following 1985, is converted to [1986-01-01T00:00, 1989-01-01T00:00). Note that there may be more than one interval associated with a single annotation, as in the *Saturdays since March 6* example. Once all annotations have been converted into intervals along the timeline, we can measure how much the intervals of different annotations overlap.

Given two sets of intervals, we define the interval precision, P_{int}, as the total length of the intervals in common between the two sets, divided by the total length of the intervals in the first set. Interval recall, R_{int} is defined as the total length of the intervals in common between the two sets, divided by the total length of the intervals in the second set. Formally:

$$I_S \bigcap I_H = \{i \cap j : i \in I_S \wedge j \in I_H\}$$

$$P_{\text{int}}(I_S, I_H) = \frac{\sum_{i \in \text{COMPACT}(I_S \bigcap I_H)} |i|}{\sum_{i \in I_S} |i|}$$

$$R_{\text{int}}(I_S, I_H) = \frac{\sum_{i \in \text{COMPACT}(I_S \bigcap I_H)} |i|}{\sum_{i \in \cup I_H} |i|}$$

where I_S and I_H are sets of intervals, $i \cap j$ is possibly the empty interval in common between the intervals i and j, $|i|$ is the length of the interval i, and COMPACT takes a set of intervals and merges any overlapping intervals.

Given two sets of annotations (e.g., one each from two time normalization systems), we define the overall precision, P, as the average of interval precisions where each annotation from the first set is paired with all annotations that textually overlap it in the second set. Overall recall is defined as the average of interval recalls where each annotation from the second set is paired with all annotations that textually overlap it in the first set. Formally:

$$\text{OI}_a(B) = \bigcup_{b \in B : \text{OVERLAPS}(a, b)} \text{INTERVALS}(b)$$

$$P(S, H) = \frac{1}{|S|} \sum_{s \in S} P_{int}(\text{INTERVALS}(s), \text{OI}_s(H))$$

$$R(S, H) = \frac{1}{|H|} \sum_{h \in H} R_{int}(\text{INTERVALS}(h), \text{OI}_h(S))$$

where S and H are sets of annotations, INTERVALS(x) gives the time intervals associated with the annotation x, and OVERLAPS (a, b) decide whether the annotations a and b share at least one character of text in common.

It is important to note that these metrics can be applied only to time expressions that yield bounded intervals. Time expressions that refer to intervals with undefined boundaries are out of the scope, like in “it takes just a minute” or “I work every Saturday”.

4 Data analysis

4.1 TimeML vs. SCATE

Both TimeML and SCATE annotations are available on a subset of the TempEval 2013 corpus (UzZaman et al., 2013) that contains a collection of news articles from different sources, such as Wall Street Journal, New York Times, Cable News Network, and Voices of America. Table 1 shows the statistics of the data. Documents from the AQUAINT and TimeBank form the training and development dataset. The SCATE corpus contains 2604 time entities (individual components of a time expression, such as *every, month, last, Monday,* etc.) annotated in the train+dev set (i.e. AQUAINT+TimeBank). These entities compose a total of 1038 time expressions (*every month, last Monday,* etc.) of which 580 yield bounded intervals, i.e. intervals with a specified start and ending (*last Monday* is bounded, while *every month* is not).

We apply the interval-based evaluation metric introduced in Section 3 to the AQUAINT and TimeBank datasets, treating the TimeML annotations as the system (S) annotator and the SCATE annotations as the human (H) annotator. Table 2 shows that the SCATE annotations cover different time intervals than the TimeML annotations. In the first row, we see that TimeML has a recall of only 92% of the time intervals identified by SCATE in the AQUAINT corpus and of only 83% in the TimeBank corpus. We manually analyzed all places where TimeML and SCATE annotations differed and found that the SCATE interpretation was always the correct one.

For example, a common case where TimeML and SCATE annotations overlap, but are not identical, is time expressions preceded by a preposition like “since”. The TimeML annotation for “Since 1985” (with a DCT of 1998-03-01T14:11) only covers the year, “1985”, resulting in the time interval [1985-01-01T00:00,1986-01-01T00:00). The SCATE annotation represents the full expression and, consequently, produces the correct time interval [1986-01-01T00:00,1998-03-01T14:11).

Another common case of disagreement is where TimeML failed to compose all pieces of a complex expression. The TimeML annotation for “10:35 a.m. (0735 GMT) Friday” annotates two separate intervals, the time and the day (and ignores “0735 GMT” entirely). The SCATE annotation recognizes this as a description of a single time interval, [1998-08-07T10:35, 1998-08-07T10:36).

TimeML and SCATE annotations also differ in how references to particular past periods are interpreted. For example, TimeML assumes that “last year” and “a year ago” have identical semantics, referring to the most recent calendar year, e.g., if the DCT is 1998-03-04, then they both refer to the interval [1997-01-01T00:00,1998-01-01T00:00). SCATE has the same semantics for “last year”, but recognizes that “a year ago” has different semantics: a period centered at one year prior to the DCT. Under SCATE, “a year ago” refers to the interval [1996-09-03T00:00,1997-09-03T00:00).

Beyond these differences in interpretation, we also observed that, while the SCATE corpus annotates time expressions anywhere in the document (including in metadata), the

TimeBank TIMEX3 annotations are restricted to the main text of the documents. The second row of Table 2 shows the evaluation when comparing overall text in the document, not just the body text. Unsurprisingly, TimeML has a lower recall of the time intervals from the SCATE annotations under this evaluation.

4.2 Types of SCATE annotations

Studying the training and development portion of the dataset, we noticed that the SCATE annotations can be usefully divided into three categories: non-operators, explicit operators, and implicit operators. We define non-operators as Numbers, Periods (e.g., *three months),* explicit intervals (e.g., Years like *1989*), and repeating intervals (Day-Of-Weeks like *Friday,* Month-of-Years like *January,* etc.). Non-operators are basically atomic; they can be interpreted without having to refer to other annotations. Operators are not atomic; they can only be interpreted with respect to other annotations they link to. For example, the THIS operator in Figure 1 can only be interpreted by first interpreting the Day-of-Week non-operator and the Between operator that it links to. We split operators into two types: explicit and implicit. We define an operator as explicit if it does not overlap with any other annotation. This occurs, for example, when the time connective *since* evokes the Between operator in Figure 1. An operator is considered to be implicit if it overlaps with another annotation. This occurs, for example, with the LAST operator in Figure 1, where *March* implies *last March,* but there is no explicit signal in the text, and it must be inferred from context.

We study how these annotation groups distribute in the AQUAINT and TimeBank documents. Table 3 shows that non-operators are much more frequent than operators (both explicit and implicit).

5 Models

We decompose the normalization of time expressions into two subtasks: a) *time entity identification* which detects the spans of characters that belong to each time expression and labels them with their corresponding time entity; and b) *time entity composition* that links relevant entities together while respecting the entity type constraints imposed by the SCATE schema. These two tasks are run sequentially using the output of the former as input to the latter. Once identification and composition steps are completed we can use the final product, i.e. semantic compositional of time entities, to feed the SCATE interpreter[1] and encode time intervals.

5.1 Time entity identification

Time entity identification is a type of sequence tagging task where each piece of a time expression is assigned a label that identifies the time entity that it evokes. We express such labels using the BIO tagging system, where B stands for the beginning of an annotation, I for the inside, and O for outside any annotation. Differing somewhat from standard sequence tagging tasks, the SCATE schema allows multiple annotations over the same span of text

[1]https://github.com/clulab/timenorm

(e.g., “Saturdays” in Figure 1 is both a DAY-OF-WEEK and a THIS), so entity identification models must be able to handle such multi-label classification.

5.1.1 Neural architectures—

Recurrent neural networks (RNN) are the state-of-the-art on sequence tagging tasks (Lample et al., 2016a; Graves et al., 2013; Plank et al., 2016) thanks to their ability to maintain a memory of the sequence as they read it and make predictions conditioned on long distance features, so we also adopt them here. We introduce three RNN architectures that share a similar internal structure, but differ in how they represent the output. They convert the input into features that feed an embedding layer. The embedded feature vectors are then fed into two stacked bidirectional Gated Recurrent Units (GRUs), and the second GRU followed by an activation function, outputs one BIO tag for each input. We select GRU for our models as they can outperform another popular recurrent unit LSTM (Long Short Term Memory), in terms of parameter updates and convergence in CPU time with the same number of parameters (Chung et al., 2014).

Our 1-Sigmoid model (Figure 2) approaches the task as a multi-label classification problem, with a set of sigmoids for each output that allow zero or more BIO labels to be predicted simultaneously. This is the standard way of encoding multi-label classification problems for neural networks, but in our experiments, we found that these models perform poorly since they can overproduce labels for each input, e.g., *03* could be labeled with both DAY-OF-MONTH and MONTH-OF-YEAR at the same time.

Our 2-Softmax model (Figure 3) splits the output space of labels into two sets: non-operators and operators (as defined in Section 4.2). It is very unlikely that any piece of text will be annotated with more than one non-operator or with more than one operator,[2] though it is common for text to be annotated with one non-operator and one operator (see Figure 1). As a result, we can use two softmaxes, one for non-operators and one for operators, and the 2-Softmax model thus can produce 0, 1, or 2 labels per input. We share input and embedding layers, but associate a separate set of stacked Bi-GRUs with each output category, as shown in Figure 3.[3]

Our 3-Softmax further splits operators into explicit operators and implicit operators (again, as defined in Section 4.2). We expect this to help the model since the learning task is very different for these two cases: with explicit operators, the model just has to memorize which phrases evoke which operators, while with implicit operators, the model has to learn to infer an operator from context (verb tense, etc.). We use three softmaxes, one each for non-operators, explicit operators, and implicit operators, and, as with 2-Softmax, we share input and embedding layers, but associate a separate set of stacked Bi-GRUs with each output category. The model looks similar to Figure 3, but with three output groups instead of two.

We feed three features as input to the RNNs:

[2]In the training data, only 4 of 1217 non-operators overlap with another non-operator, and only 6 of 406 operators overlap with another operator. For example, a *NYT said in an editorial on Saturday, April 25*, *Saturday* is labeled as [DAY-OF-WEEK, LAST, INTERSECTION] where the last two labels are operators.

[3]In preliminary experiments, we tried sharing GRU layers as well, but this generally resulted in worse performance.

Text: The input word itself for the word-by-word model, or a the single input character for the character-by-character model.

Unicode character categories: The category of each character as defined by the Unicode standard.[4] This encodes information like the presence of uppercase (Lu) or lowercase (Ll) letters, punctuation (Po), digits (Nd), etc. For the word-by-word model, we concatenate the character categories of all characters in the word (e.g., *May* becomes LuLlLl).

Part-of-speech: The part-of-speech as determined by the Stanford POS tagger (Toutanova et al., 2003). We expect this to be useful for, e.g., finding verb tense to help distinguish between implicit Last and Next operators. For the character-by-character model, we repeat the word-level part-of-speech tag for each character in the word, and characters with no part-of-speech (e.g., spaces) get no tag.

5.1.2 Input: words vs. characters—

Identifying SCATE-style time entity is a sequence tagging task, similar to named entity recognition (NER), so we take inspiration from recent work in neural architectures for NER. The first neural NER models followed the prior (non-neural) work in approaching NER as a word classification problem, applying architectures such as sliding-window feedforward neural networks (Qi et al., 2009), convolutional neural networks (CNNs) with conditional random field (CRF) layers (Collobert et al., 2011), and LSTM with CRF layers and hand-crafted features (Huang et al., 2015). More recently, character-level neural networks have also been proposed for NER, including several which combine a CNN or LSTM for learning character-based representations of words with an LSTM or LSTM-CRF for word-by-word labeling (Chiu and Nichols, 2016; Lample et al., 2016b; Ma and Hovy, 2016), as well as character-by-character sequence-to-sequence networks (Gillick et al., 2016; Kuru et al., 2016).

Based on these works, we consider two forms of input processing for our RNNs: word-by-word vs. character-by-character. Several aspects of the time normalization problem make the character-based approach especially appealing. First, many time phrases involve numbers that must be interpreted semantically (e.g., a good model should learn that months cannot be a number higher than 12), and digit-bydigit processing of numbers allows such interpretations, while treating each number as a word would result in a sparse, intractable learning problem. Second, word-based models assume that we know how to tokenize the text into words, but at times present challenging formats such as *overnight,* where *over* evokes a Last operator and *night* is a Part-of-Day. Finally, character-based models can ameliorate out-of-vocabulary (OOV) words, which are a common problem when training sparse datasets. (Hybrid word-character models, such as the LSTM-CNNs-CRF (Ma and Hovy, 2016) can address this last problem, but not the previous two.)

For our word-based model, we apply the NLTK Tokenizer (Bird et al., 2009) to each sentence. We further tokenize with the regular expression “\d+|[^\d\W] + |\S” to break apart alphanumeric expressions like *1620EDT*. However, the tokenizer is unable to break-apart

[4]See http://unicode.org/notes/tn36/

expressions such as *19980206* and *overnight*. For our character-based model, no tokenization is applied and every character (including whitespace characters) is fed as input.

5.2 Time entity composition

Once the entities of the time-expressions are identified, they must be composed in order to obtain their semantic interpretation. This step of the analysis consists of two parts: linking the entities that make up a time-expression together and completing the entities' properties with the proper values. For both cases, we set a simple set of rules that follow the constraints imposed by the SCATE schema[5].

5.2.1 Time entity linking—Algorithm 1 shows the process followed to obtain the links between the time entities. First, we define an empty stack that will store the entities belonging to the same time expression. Then, we iterate over the list of entities of a document sorted by their starting character offsets (SortByStart). For each of these entities (*entity*$_1$) and for each entity in the stack (*entity*$_2$), we check if the guidelines specify a possible link (LinkIsValid) between the types of *entity*$_1$ and *entity*$_2$. If such a link is possible, and it has not already been filled by another annotation, we greedily make the link (CREATELINK). When the distance in the number of characters between the entity and the end of the stack is bigger than 10, we assume that the entities do not belong to the time expression. Thus, we empty the stack.[6]

Algorithm 1

Linking time entities

```
stack = ∅
for entity_1 in SORTBYSTART(entities) do
  if START(entity_1) - END(stack) > 10 then stack = ∅
  end if
  for entity_2 in stack do
    if LINKISVALID(entity_1, entity_2) then CREATELINK(entity_1, entity_2)
    end if
  end for
  PUSH(stack, entity_1)
end for
```

For example, our time entity identification model gets the Year, Month-of-Year and Day-Of-Month for the time-expression *1992-12-23*. Our time entity composition algorithm then iterates over these entities. At the beginning the stack is empty, it just pushes the entity *1992* (Year) into the stack. For the entity *12* (Month-of-Year) it checks if the guidelines define a possible link between this entity type and the one currently in the stack (Year). In this case, the guidelines establish that a Year can have a Sub-interval link to a Season-of-Year, a Month-of-Year or Week-of-Year. Thus, the algorithm creates a Sub-interval link between

[5] https://github.com/bethard/anafora-annotations/blob/master/.schema/timenorm-schema.xml

[6] The distance threshold was selected based on the performance on the development dataset.

1992 and *12*. The entity *12* is then pushed into the stack. This process is repeated for the entity *23* (Day-Of-Month) checking if there was a possible link to the entities in the stack (*1992, 12*). The guidelines define a possible Sub-interval link between Month-of-Year and Day-Of-Month, so a link is created here as well. Now, suppose that the following time entity in the list is several words ahead of *23* so the character distance between both entities is larger than 10. If that is the case the stack is empty and the process starts again to compose a new time expression.

5.2.2 Property completion—

The last step is to associate each time entity of a time-expression with a set of properties that include information needed for its interpretation. Our system decides the value of these properties as follows:

Type: The SCATE schema defines that some entities can only have specific values. For example, a Season-of-Year can only be Spring, Summer, Fall or Winter, a Month-of-Year can only be January, February, March, etc. To complete this property we take the text span of the time entity and normalize it to the values accepted in the schema. For example, if the span of a Month-of-Year entity was the numeric value *01* we would normalize it to January, if its span was *Sep.* we would normalize it to September, and so on.

Value: This property contains the value of a numerical entity, like Day-of-Month or Hour-of-Day. To complete it, we just take the text span of the entity and convert it to an integer. If it is written in words instead of digits (e.g., *nineteen* instead of 19), we apply a simple grammar[7] to convert to an integer.

Semantics: In news-style texts, it is common that expressions like *last Friday*, when the DCT is a Friday, refer to the day as the DCT instead of the previous occurrence (as it would in more standard usage of *last*). SCATE indicates this with the Semantics property, where the value Interval-Included indicates that the current interval is included when calculating the last or next occurrence. For the rest of the cases the value Interval-Not-Included is used. In our system, when a LAST operator is found, if it is linked to a Day-of-Week (e.g. *Friday*) that matches the DCT, we set the value of this property to Interval-Included.

Interval-Type: Operators like Next or Last need an interval as reference in order to be interpreted. Normally, this reference is the DCT. For example, *next week* refers to the week following the DCT, and in such a case the value of the property Interval-Type for the operator Next would be DocTime. However, sometimes the operator is linked to an interval that serves as reference by itself, for example, "by the year 2000". In this cases the value of the Interval-Type is Link. Our system sets the value of this property to Link if the operator is linked to a Year and DocTime otherwise. This is a very coarse heuristic; finding the proper anchor for a time expression is a challenging open problem for which future research is needed.

[7]https://github.com/ghewgill/text2num

5.3 Automatically generated training data

Every document in the dataset starts with a document creation time. These time expressions are quite particular; they occur in isolation and not within the context of a sentence and they always yield a bounded interval. Thus their identification is a critical factor in an interval based evaluation metric. However, document times appear in many different formats: “Monday, July-24, 2017”, “07/24/17 09:52 AM”, “08-15-17 1337 PM”, etc. Many of these formats are not covered in the training data, which is drawn from a small number of news sources, each of which uses only a single format. We therefore designed a time generator to randomly generate an extra 800 isolated training examples for a wide variety of such expression formats. The generator covers 33 different formats[8] which include variants covering abbreviation, with/without delimiters, mixture of digits and strings, and different sequences of time units.

6 Experiments

We train and evaluate our models on the SCATE corpus described in Section 4. As a development dataset, 14 documents are taken as a random stratified sample from the TempEval 2013 (TimeBank + AQUAINT) portion shown in Table 1, including broadcast news documents (1 ABC, 1 CNN, 1 PRI, 1 VOA), and newswire documents (5 AP, 1 NYT, 4 WSJ). We use the interval-based evaluation metric described in Section 3, but also report more traditional information extraction metrics (precision, recall, and F_1) for the time entity identification and composition steps. Let S be the set of items predicted by the system and H is the set of items produced by the humans, precision (P), recall (R), and F_1 are defined as:

$$P(S, H) = \frac{|S \cap H|}{|S|}$$

$$R(S, H) = \frac{|S \cap H|}{|H|}$$

$$F_1(S, H) = \frac{2 \cdot P(S, H) \cdot R(S, H)}{P(S, H) + R(S, H)}.$$

For these calculations, each item is an annotation, and one annotation is considered equal to another if it has the same character span (offsets), type, and properties (with the definition applying recursively for properties that point to other annotations).

To make the experiments with different neural architectures comparable, we tuned the parameters of all models to achieve the best performance on the development data. Due to space constraints, we only list here the hyper-parameters for our best Char 3-Softmax: the embedding size of the character-level text, word-level text, POS tag, and unicode character category features are 128, 300, 32 and 64, respectively. To avoid overfitting, we used

[8]We use the common formats available in office suites, specifically, LibreOffice.

dropout with probabilities 0.25, 0.15 and 0.15 for the 3 features, respectively; the sizes of the first and second layer GRU units are set as 256 and 150. We trained the model with RMSProp optimization on mini-batches of size 120, and followed standard recommendations to leave the optimizer hyperparameter settings at their default values. Each model is trained for at most 800 epochs, the longest training time for Char 3-Softmax model is around 22 hours using 2x NVIDIA Kepler K20X GPU.

6.1 Model selection

We compare the different time entity identification models described in Section 5.1, training them on the training data, and evaluating them on the development data. Among the epochs of each model, we select the epoch based on the output(s) which the model is good at predicting because based on its weakness, the model would yield unstable results in our preliminary experiments. For example, for 3-Softmax models, our selections rely on the performances of non-operators and implicit-operators. Table 4 shows the results of the development phase.

First, we find that the character-based models outperform the word-based models.[9] For example, the best character-based model achieves the F_1 of 81.7 (Char 3-Softmax),which is significantly better than the best word-based model achieving the F_1 of only 66.6 (p=0).[10] Second, we find that Softmax models outperform Sigmoid models. For example, the Char 3-Softmax model achives the F_1 of 81.7, significantly better than 56.4 F_1 of the Char 1-Sigmoid model (p=0). Third, for both character- and word-based models, we find that 3-Softmax significantly outperforms 2-Softmax: the Char 3-Softmax F_1 of 81.7 is better than the Char 2-Softmax F_1 of 73.6 (p=0) and the Word 3-Softmax F_1 of 66.6 is better than the Word 2-Softmax F_1 of 61.2 (p=0.0254). Additionally, we find that all models are better at identifying non-operators than operators and that the explicit operators are the hardest to solve. For example, the Char 3-Softmax model gets 92.4 F_1 for non-operators, 36.1 F_1 for explicit operators and 79.1 F_1 for implicit operators. Finally, we also train the best model, Char 3-Softmax, using the generated annotations described in Section 5.3 and achieve 76.8 F_1 (Char 3-Softmax extra), i.e., the model performs better without the extra data (p=0). This is probably a result of overfitting due to the small variety of time formats in the training and development data.

From this analysis on the development set, we select two variants of the Char 3-softmax architecture for evaluation on the test set: Char 3-Softmax and Char 3-Softmax extra. These models were then coupled with the rule-based linking system described in Section 5.2 to produce a complete SCATE-style parsing system.

6.2 Model evaluation

We evaluate both Char 3-Softmax and Char 3-Softmax extra on the test set for identification and composition tasks. Table 5 shows the results. On the identification task, Char 3-Softmax extra is no worse than using the original dataset with the overall F_1 61.5 vs. 61.3 (p=0.5899),

[9]We briefly explored using pre-trained word embeddings to try to improve the performance of the Word 1-Sigmoid model, but it yielded a performance that was still worse than the character-based model, so we didn't explore it further.
[10]We used a paired bootstrap resampling significance test.

and using extra generated data the model is better at predicting non-operators and implicit operators with higher precisions (p=0.0096), which is the key to produce correct bounded time intervals.

To compare our approach with the state-of-the-art, we run HeidelTime on the test documents and make use of the metric described in Section 3. This way, we can compare the intervals produced by both systems no matter the annotation schema. Table 6 shows that our model with additional randomly generated training data outperforms HeidelTime in terms of Precision, with a significant difference of 12.6 percentage points (p=0.011), while HeidelTime obtains a non-significant better performance in terms of Recall (p=0.1826). Overall, our model gets 3.3 more percentage points than HeidelTime in terms of $F\mathbf{1}$ (p=0.2485). Notice that, although the model trained without extra annotations is better in time entity composition (see Table 5), it performs much worse at producing final intervals. This is caused by the fact that this model fails to identify the non-operators that compound dates in unseen formats (see Section 5.3).

However, evaluating HeidelTime in the SCATE annotations may not be totally fair. HeidelTime was developed following the TimeML schema and, as we show in Section 4, SCATE covers a wider set of time expressions. For this reason, we perform an additional evaluation. First, we compare the annotations in the test set using our interval-based metric, similar to the comparison reported in Table 2, and select those cases where TimeML and SCATE match perfectly. Then, we remove the rest of the cases from the test set. Consequently, we also remove the predictions given by the systems, both ours and HeidelTime, for those instances. Finally, we run the interval scorer using the new configuration. As can be seen in Table 7 all the models improve their performances. However, our model still performs better when it is trained with the extra annotations.

The SCATE interpreter that encodes the time intervals needs the compositional graph of a time-expression to have all its elements correct. Thus, failing in the identification of any entity of a time-expression results in totally uninterpretable graphs. For example, in the expression *next year,* if our model identifies *year* as a Period instead of an Interval it cannot be linked to *next* because it violates the SCATE schema. The model can also fail in the recognition of some time-entities, like *summer* in the expression *last summer*. This identification errors are caused mainly by the sparse training data. As graphs containing these errors produce unsolvable logical formulae, the interpreter cannot produce intervals and hence the recall decreases. Within those intervals that are ultimately generated, the most common mistake is to confuse the Last and Next operators, and as a result an incorrectly placed interval even with correctly identified non-operators. For example, if an *October* with an implicit NEXT operator is instead given a LAST operator, instead of referring to [2013-10-01T00:00,2013-11-01T00:00), it will refer to [2012-10-01T00:00, 2012-11-01T00:00). Missing implicit operators is also the main source of errors for HeidelTime, which fails with complex compositional graphs. For example, *that January day in 2011* is annotated by HeidelTime as two different intervals, corresponding respectively to *January* and *2011*. As a consequence, HeidelTime predicts not one but two incorrect intervals, affecting its precision.

7 Discussion

As for the time entity identification task, the performance differences between development and test dataset could be attributed to the annotation distributions of the datasets. For example, there are 10 Season-Of-Year annotations in the test set while there are no such annotations in the development dataset; the relative frequencies of the annotations Minute-Of-Hour, Hour-Of-Day, Two-Digit-Year and Time-Zone in the test set are much lower, and our models are good at predicting such annotations. Explicit operators are very lexically-dependent, e.g. LAST corresponds to one word from the set {*last*, *latest*, *previously*, *recently*, *past*, *over*, *recent*, *earlier*, *the past*, *before*}, and the majority of them appear once or twice in the training and development sets.

Our experiments verify the advantages of character-based-models in predicting SCATE annotations, which are in agreement with our explanations in Section 5.1.2: word-based-models tend to fail to distinguish numbers from digit-based time expressions. It's difficult for word-based-models to catch some patterns of time expressions, such as *24th* and *25th, August* and *Aug.,* etc., while character-based models are robust to such variance. We ran an experiment to see whether these benefits were unique to compositional annotations like those of SCATE, or more generally to simply recognizing time expressions. We used the TimeML annotations from AQUAINT and TimeBank (see Table 1) to train two multi-class classifiers to identify TIMEX3 annotations. The models were similar to our Char 3-Softmax and Word 3-Softmax models, using the same parameter settings, but with a single softmax output layer to predict the four types of TIMEX3: DATE, TIME, DURATION, and SET. As shown in Table 8,on the test set the word-based model significantly outperforms the character-based model in terms of both time expressions (p=0.0428) and the subset of time expressions that contain digits (p=0.0007). These results suggest that the reason character-based models are more successful on the SCATE annotations is that SCATE breaks time expressions down into meaningful sub-components. For example, TimeML would simply call *Monday, 1992-05-04* a DATE, and call *15:00:00 GMT Saturday* a TIME. SCATE would identify four and five, respectively, different types of semantic entities in these expression; and each SCATE entity would be either all letters or all digits. In TimeML, the model is faced with difficult learning tasks, e.g., that sometimes a weekday name is part of a DATE and sometimes it is part of a TIME, while in SCATE, a weekday name is always a DAY-OF-WEEK.

On the other hand, running the entity composition step with gold entity identification achieves 72.6 in terms of F1. One of the main causes of errors in this step is the heuristic to complete the INTERVAL-TYPE property. As we explain in Section 5.2, we implement a too coarse set of rules for this case. Another source of errors is the distance of the 10 characters we use to decide if the time entities belong to the same time expression. This condition prevents the creation of some links, for example, the expression "Later" at the beginning of a sentence typically refers to another time interval in a previous sentence, so the distance between them is much longer.

8 Conclusion

We have presented the first model for time normalization trained on SCATE-style annotations. The model outperforms the rule-based state-of-the-art, proving that describing time expressions in terms of compositional time entities is suitable for machine learning approaches. This broadens the research in time normalization beyond the more restricted TimeML schema. We have shown that a character-based neural network architecture has advantages for the task over a word-based system, and that a multi-output network performs better than producing a single output. Furthermore, we have defined a new interval-based evaluation metric that allows us to perform a comparison between annotations based on both SCATE and TimeML schema, and found that SCATE provides a wider variety of time expressions. Finally, we have seen that the sparse training set available induces model overfitting and that the largest number of errors are committed in those cases that appear less frequently in the annotations. This is more significant in the case of explicit operators because they are very dependent on the lexicon. Improving performance on these cases is our main goal for future work. According to the results presented in this work, it seems that a solution would be to obtain a wider training set, so a promising research line is to extend our approach to automatically generate new annotations.

9 Software

The code for the SCATE-style time normalization models introduced in this paper is available at https://github.com/clulab/timenorm.

Acknowledgements

We thank the anonymous reviewers as well as the action editor, Mona Diab, for helpful comments on an earlier draft of this paper. The work was funded by the THYME project (R01LM010090) from the National Library Of Medicine, and used computing resources supported by the National Science Foundation under Grant No. 1228509. The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Library Of Medicine, National Institutes of Health, or National Science Foundation.

References

Bethard Steven and Parker Jonathan. 2016 A semantically compositional annotation scheme for timenormalization. In Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016), Paris, France, 5 European Language Resources Association (ELRA).

Bethard Steven. 2013 A synchronous con text free grammar for time normalization. In Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, pages 821–826, Seattle, Washington, USA, 10 Association for Computational Linguistics.

Bird Steven, Klein Ewan, and Loper Edward. 2009 Natural language processing with Python: analyzing text with the natural language toolkit. O'Reilly Media, Inc.

Chiu Jason P. C. and Nichols Eric. 2016 Named Entity Recognition with Bidirectional LSTM-CNNs. Transactions of the Association for Computational Linguistic, 4:357–370.

Chung Junyoung, Gulcehre Caglar, Cho KyungHyun, and Bengio Yoshua. 2014 Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555v1.

Collobert Ronan, Weston Jason, Bottou Léon, Karlen Michael, Kavukcuoglu Koray, and Kuksa Pavel. 2011 Natural language processing (almost) from scratch. The Journal of Machine Learning Research, 12:2493–2537, 11.

Fischer Frank and Strötgen Jannik. 2015 When Does (German) Literature Take Place? On the Analysis of Temporal Expressions in Large Corpora. In Proceedings of DH 2015: Annual Conference of the Alliance of Digital Humanities Organizations, volume 6, Sydney, Australia.

Gillick Dan, Brunk Cliff, Vinyals Oriol, and Subramanya Amarnag. 2016 Multilingual language processing from bytes. In Knight Kevin, Nenkova Ani, and Rambow Owen, editors, NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016, pages 1296–1306. The Association for Computational Linguistics.

Graves Alex, Mohamed Abdel-rahman, and Hinton Geoffrey. 2013 Speech recognition with deep recurrent neural networks. In 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, pages 6645–6649. IEEE.

Han Benjamin and Lavie Alon. 2004 A framework for resolution of time in natural language. 3(1):11–32, 3.

Huang Zhiheng, Xu Wei, and Yu Kai. 2015 Bidirectional LSTM-CRF models for sequence tagging. CoRR, abs/1508.01991.

Kuru Onur, Can Ozan Arkan, and Yuret Deniz. 2016 Charner: Character-level named entity recognition. In COLING 2016, 26th International Conference on Computational Linguistics, Proceedings of the Conference: Technical Papers, December 11-16, 2016, Osaka, Japan, pages 911–921.

Lample Guillaume, Ballesteros Miguel, Subramanian Sandeep, Kawakami Kazuya, and Dyer Chris. 2016a Neural architectures for named entity recognition. In Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 260–270. Association for Computational Linguistics.

Lample Guillaume, Ballesteros Miguel, Subramanian Sandeep, Kawakami Kazuya, and Dyer Chris. 2016b Neural architectures for named entity recognition. In NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016, pages 260–270.

Lee Kenton, Artzi Yoav, Dodge Jesse, and Zettlemoyer Luke. 2014 Context-dependent semantic parsing for time expressions. In Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 1437–1447, Baltimore, Maryland, 6 Association for Computational Linguistics.

Lin Chen, Karlson Elizabeth W., Dligach Dmitriy, Ramirez Monica P., Miller Timothy A., Mo Huan, Braggs Natalie S., Cagan Andrew, Gainer Vivian S., Denny Joshua C., and Savova Guergana K.. 2015 Automatic identification of methotrexate-induced liver toxicity in patients with rheumatoid arthritis from the electronic medical record. Journal of the American Medical Informatics Association, 22(e1):e151–e161. [PubMed: 25344930]

Llorens Hector, Derczynski Leon, Gaizauskas Robert J., and Saquete Estela. 2012 TIMEN: An Open Temporal Expression Normalisation Resource. In Language Resources and Evaluation Conference, pages 3044–3051. European Language Resources Association (ELRA).

Ma Xuezhe and Hovy Eduard. 2016 End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF. In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL 2016), volume 1 Association for Computational Linguistics.

Mazur Pawet and Dale Robert. 2010 Wikiwars: A new corpus for research on temporal expressions. In Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing, EMNLP ‘10, pages 913–922, Stroudsburg, PA, USA Association for Computational Linguistics.

Plank Barbara, Søgaard Anders, and Goldberg Yoav. 2016 Multilingual part-of-speech tagging with bidirectional long short-term memory models and auxiliary loss. In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), pages 412–418, Berlin, Germany, 8 Association for Computational Linguistics.

Pustejovsky James, Castaño José, Ingria Robert, Saurí Roser, Gaizauskas Robert, Setzer Andrea, and Katz Graham. 2003a TimeML: Robust Specification of Event and Temporal Expressions in Text. In IWCS-5, Fifth International Workshop on Computational Semantics.

Pustejovsky James, Hanks Patrick, Sauri Roser, See Andrew, Gaizauskas Robert, Setzer Andrea, Radev Dragomir, Sundheim Beth, Day David, Ferro Lisa, and Lazo Marcia. 2003b The TimeBank corpus. In Proceedings of Corpus Linguistics 2003, Lancaster.

Pustejovsky James, Lee Kiyong, Bunt Harry, and Romary Laurent. 2010 ISO-TimeML: An International Standard for Semantic Annotation. In Proceedings of the 7th International Conference on Language Resources and Evaluation (LREC'10), Valletta, Malta European Language Resources Association (ELRA).

Qi Yanjun, Kavukcuoglu Koray, Collobert Ronan, Weston Jason, and Kuksa Pavel P.. 2009 Combining labeled and unlabeled data with word-class distribution learning. In Proceedings of the 18th ACM conference on Information and knowledge management, ACM, pages 1737–1740.

Schilder Frank. 2004 Extracting meaning from temporal nouns and temporal prepositions. ACM Transactions on Asian Language Information Processing (TALIP) - Special Issue on Temporal Information Processing, 3(1):33–50, 3.

Strötgen Jannik and Gertz Michael. 2013 Multilingual and cross-domain temporal tagging. Language Resources and Evaluation, 47(2):269–298.

Strötgen Jannik and Gertz Michael. 2015 A baseline temporal tagger for all languages. In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pages 541–547, Lisbon, Portugal, 9 Association for Computational Linguistics.

Strötgen Jannik, Zell Julian, and Gertz Michael. 2013 Heideltime: Tuning English and developing Spanish resources for TempEval-3. In Proceedings of the Seventh International Workshop on Semantic Evaluation, SemEval '13, pages 15–19. Association for Computational Linguistics.

Toutanova Kristina, Klein Dan, Manning Christopher D., and Singer Yoram. 2003 Feature-rich part-of-speech tagging with a cyclic dependency network. In Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1, NAACL '03 , pages 173–180, Stroudsburg, PA, USA Association for Computational Linguistics.

UzZaman Naushad, Llorens Hector, Derczynski Leon, Allen James, Verhagen Marc, and Pustejovsky James. 2013 SemEval-2013 Task 1: TempEval-3: Evaluating Time Expressions, Events, and Temporal Relations. In Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013), pages 1–9, Atlanta, Georgia, USA, 6 Association for Computational Linguistics.

Verhagen Marc, Gaizauskas Robert, Schilder Frank, Hepple Mark, Katz Graham, and Pustejovsky James. 2007 SemEval-2007 Task 15: TempEval Temporal Relation Identification. In Proceedings of the 4th International Workshop on Semantic Evaluations, SemEval '07, pages 75–80, Prague, Czech Republic.

Verhagen Marc, Sauri Roser, Caselli Tommaso, and Pustejovsky James. 2010 SemEval-2010 Task 13: TempEval-2. In Proceedings of the 5th International Workshop on Semantic Evaluation, pages 57–62, Uppsala, Sweden, 7 Association for Computational Linguistics.

Vossen Piek, Agerri Rodrigo, Aldabe Itziar, Cybulska Agata, Marieke van Erp Antske Fokkens, Laparra Egoitz, Minard Anne-Lyse, Alessio Palmero Aprosio German Rigau, Rospocher Marco, and Segers Roxane. 2016 NewsReader: Using knowledge resources in a cross-lingual reading machine to generate more knowledge from massive streams of news Special Issue Knowledge-Based Systems, Elsevier.

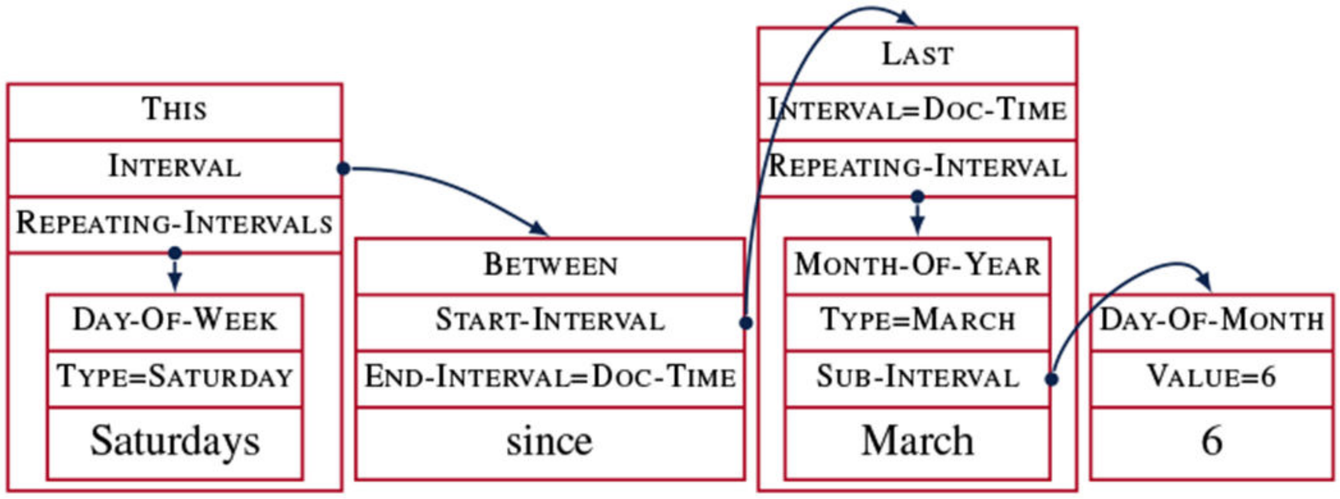

Figure 1:
Annotation of the expression *Saturdays since March 6* following the SCATE schema.

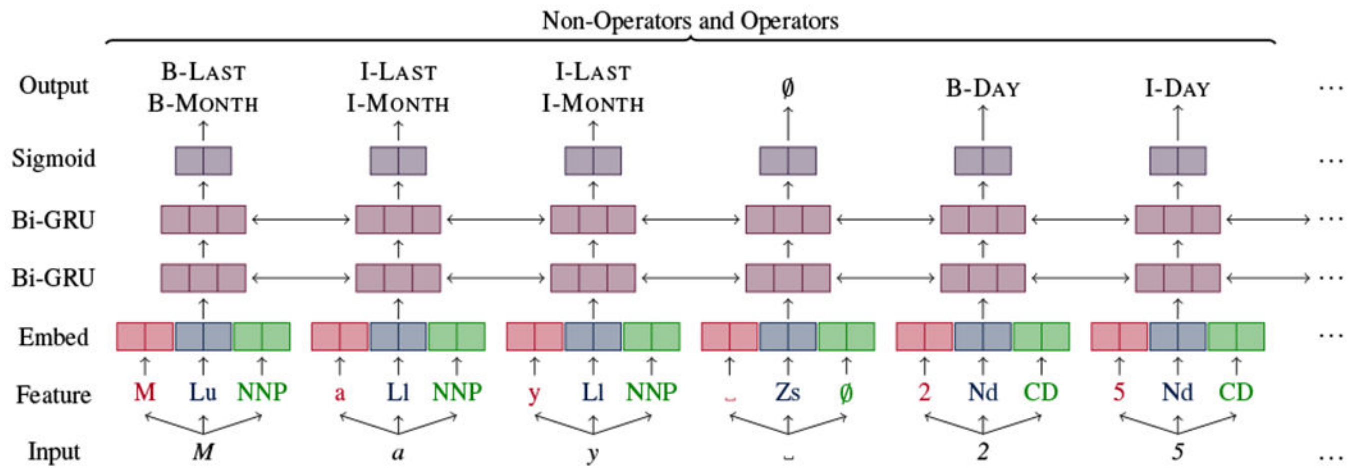

Figure 2:
Architecture of the 1-Sigmoid model. The input is *May 25*. In SCATE-style annotation, *May* is a Month-Of-Year (a non-operator), with an implicit Last (an operator) over the same span, and 25 is a Day-Of-Month. At the feature layer, *M* is an uppercase letter (Lu), *a* and *y* are lowercase letters (L1), space is a separator (Zs), and *May* is a proper noun (NNP).

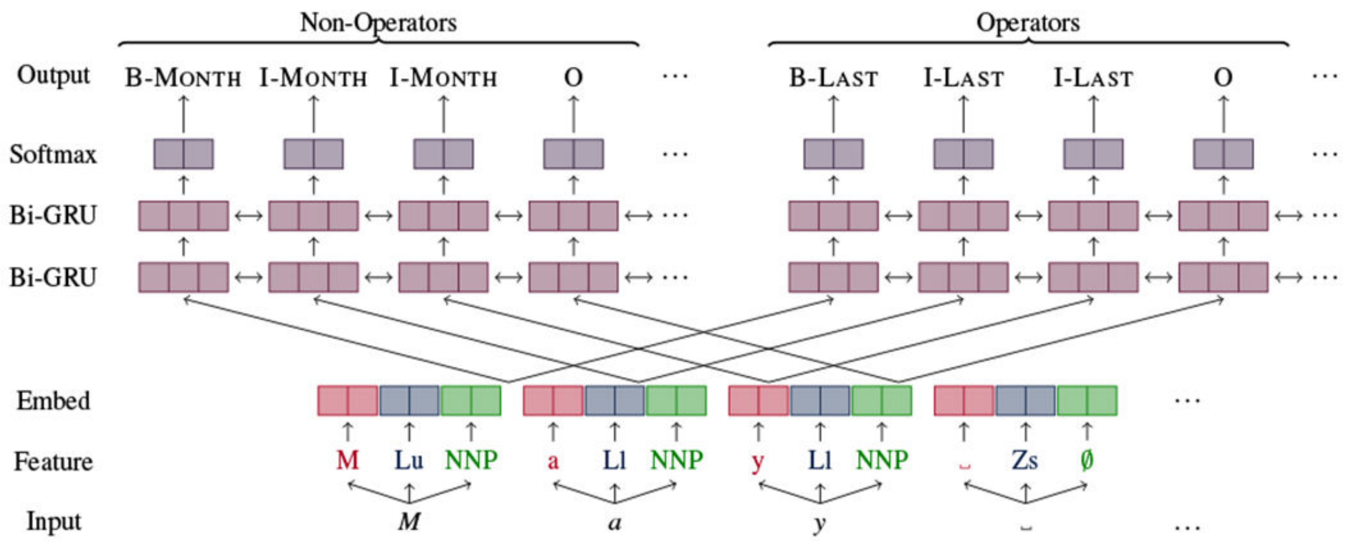

Figure 3:
Architecture of the 2-Softmax model. The input is *May*. The SCATE annotations and features are the same as in Figure 2.

Table 1:

Number of documents, TimeML TIMEX3 annotations and SCATE annotations for the subset of the TempEval 2013 corpus annotated with both schemas.

	AQUAINT	TimeBank	Test
Documents	10	68	20
Sentences	251	1429	339
TimeML timex3	61	499	158
SCATE entities	333	1810	461
SCATE time exp.	114	715	209
SCATE bounded	67	403	93

Table 2:

Comparison of TimeML and SCATE annotations.

	AQUAINT			**TimeBank**		
	P	**R**	**F1**	**P**	**R**	**F1**
Body text	92.2	92.2	92.2	82.4	83.0	82.7
All text	92.2	67.1	77.7	82.4	71.2	76.4

Table 3:

Distribution of time entity annotations in AQUAINT+TimeBank.

Non-Op	Exp-Op	Imp-Op	Total
1497	305	219	2021
74%	15%	11%	100%

Table 4:

Precision (P), recall (R), and F_1 for the different neural network architectures on *Time entity identification* on the development data.

Model	P	R	F1
Word 1-Sigmoid	60.2	52.0	55.8
Char 1-Sigmoid	54.0	59.0	56.4
Word 2-Softmax	58.7	63.9	61.2
Char 2-Softmax	74.8	72.4	73.6
Word 3-Softmax	68.3	64.9	66.6
Char 3-Softmax	88.2	76.1	81.7
Char 3-Softmax extra	80.6	73.4	76.8

Table 5:

Results on the test set for *Time entity identification* (Ident) and *Time entity composition* (Comp) steps. For the former we report the performances for each entity set: non-operators (Non-Op), explicit operators (Exp-Op) and implicit operators (Imp-Op).

	Char 3-Softmax			**Char 3-Soft. extra**		
	P	***R***	***F_1***	***P***	***R***	***F_1***
Non-Op	79.2	63.2	70.3	87.4	63.2	73.4
Exp-Op	52.6	36.6	43.2	39.8	38.7	39.3
Imp-Op	53.3	47.1	50.0	65.4	50.0	56.7
Ident	70.0	54.5	61.3	69.4	55.3	61.5
Comp	59.7	46.5	52.3	57.7	46.0	51.2

Table 6:

Precision (*P*), recall (*R*), and F_1 of our models on the test data producing bounded time intervals. For comparison, we include the results obtained by HeidelTime.

Model	***P***	***R***	F_1
HeidelTime	70.9	76.8	73.7
Char 3-Softmax	73.8	62.4	67.6
Char 3-Softmax extra	82.7	71.0	76.4

Table 7:

Precision (P), recall (R), and F_1 on bounded intervals on the *TimeML/SCATE perfect overlapping* test data.

Model	***P***	***R***	***F*$_1$**
HeidelTime	70.7	80.2	75.1
Char 3-Softmax	74.3	64.2	68.9
Char 3-Softmax extra	83.3	74.1	78.4

Table 8:

Precision *(P)*, recall *(R)*, and F_1 for character-based and word-based models in predicting TimeML TIMEX3 annotations on the TempEval 2013 test set. TIMEX3-Digits is the subset of annotations that contain digits.

	TIMEX3			**TIMEX3-Digits**		
	P	***R***	F_1	***P***	***R***	F_1
Char	70.2	62.7	66.2	73.8	71.4	72.6
Word	81.3	69.0	74.7	86.2	79.4	82.6