In Proceedings of the 30th AAAI Conference on Artificial Intelligence
(AAAI-16), Phoenix, Arizona, February 2016.

Learning Statistical Scripts with LSTM Recurrent Neural Networks

Karl Pichotta and **Raymond J. Mooney**
{pichotta,mooney}@cs.utexas.edu
Department of Computer Science
The University of Texas at Austin
Austin, TX 78712, USA

Abstract

Scripts encode knowledge of prototypical sequences of events. We describe a Recurrent Neural Network model for statistical script learning using Long Short-Term Memory, an architecture which has been demonstrated to work well on a range of Artificial Intelligence tasks. We evaluate our system on two tasks, inferring held-out events from text and inferring novel events from text, substantially outperforming prior approaches on both tasks.

Introduction

Text understanding requires commonsense inferences based on world knowledge. *Scripts*, which model stereotypical sequences of events, encode one type of useful world knowledge. A script system can infer events from text: e.g., given "After being laid off, Smith is looking for work," we would like to be able to predict that "Smith will have a job interview," and perhaps "Smith will find a job." Inferences of this type are required for robust question-answering systems.

The use of manually-written scripts dates back to the seminal work of Schank and Abelson (1977). Mooney and DeJong (1985) provide an early non-statistical method of inducing scripts automatically from text. More recently, a growing body of work has followed Chambers and Jurafsky (2008) in describing methods of automatically learning statistical models of event sequences from large text corpora.

We describe a novel statistical script model which uses a recurrent neural network with a Long Short-Term Memory (LSTM) architecture. Recently, LSTMs have yielded substantial performance gains in Machine Translation (Sutskever, Vinyals, and Le 2014) Speech Recognition (Graves, Mohamed, and Hinton 2013) Language Modeling (Sundermeyer, Schlüter, and Ney 2012) and captioning images and videos (Donahue et al. 2015; Venugopalan et al. 2015), among other tasks.

We are the first to apply LSTMs to the task of script learning, demonstrating superior performance over a number of competitive baselines, including the best published method. Our method can directly incorporate noun information about event arguments, which most previous methods of script learning for event inference do not easily admit. We evaluate our proposed LSTM script system against a number of baselines on the task of predicting held-out verbs with coreference information about their arguments, showing a 22.6% relative improvement compared to the strongest baseline. Second, we evaluate on the more difficult task of predicting held-out verbs with argument nouns, demonstrating a 64.9% relative improvement over the most competitive baseline. We find that coreference information is empirically useful for predicting noun arguments, and, conversely, noun information helps to predict argument coreference information. Third, we demonstrate that human annotators judge the top inferences made by LSTM script models to be qualitatively superior to those from baseline systems. Finally, we provide a qualitative analysis of the LSTM script model.

Background

We briefly review the field of statistical script learning and give a short description of Long Short-Term Memory.

Statistical Script Learning

Chambers and Jurafsky (2008; 2009) and Jans et al. (2012) propose script models of (verb, dependency) pairs[1] which co-occur with the same entity, capable of encoding, for example, that the grammatical subject of *kill* will frequently also occur as the direct object of *arrest*. These models are evaluated quantitatively on the *Narrative Cloze* task (Chambers and Jurafsky 2008), in which an event is held out from a document and a system is judged by its ability to infer this held-out pair from the remainder of the document.

These pair-models are incapable of expressing interactions between entities. For example, the event "X orders Y" may strongly predict the event "X eats Y," but pair-based models have no straightforward way of representing these multi-argument events or making such inferences. Motivated by these shortcomings, Pichotta and Mooney (2014) propose a model which represents events as relational atoms, like $eat(X, Y)$. They find that modeling co-occurring multi-argument events and predicting simpler pair events provides superior performance to modeling co-occurring pair events

[1] "Dependencies" here are grammatical structures output by a syntactic parser, one of *subject*, *direct object*, or *preposition*.

directly. These events do not directly incorporate noun information. Events with similar relational structure are used in Balasubramanian et al. (2013) and Modi and Titov (2014), who present systems for other tasks.

Recurrent Neural Nets and Long Short-Term Memory

Recurrent Neural Net (RNN) sequence models learn to map input sequences to output sequence via a continuous vector-valued intermediate hidden state. The most basic RNNs are difficult to train due to the so-called vanishing and exploding gradient problem. the phenomenon that the gradient signal used to train the network tends to either approach zero ("vanish") or diverge ("explode") as it is propagated back through timesteps during learning, leading to instability. Further, long-distance data dependencies, in which some timestep's input is highly predictive of a much later output, are not well modeled by simple RNNs.

Long Short-Term Memory (LSTM) units (Hochreiter and Schmidhuber 1997) are designed to obviate these problems by introducing a more complicated hidden unit which targets long-distance dependencies and addresses the vanishing gradient problem. The LSTM formulation we use is described in Zaremba and Sutskever (2014). The LSTM unit is a composition of easily differentiable functions, so we may use standard gradient-based methods to train all parameters.

LSTMs are a natural fit to script learning: some events will be highly predictive of events far ahead in the sequence (e.g. "the army invaded" may be indicative of "a treaty was signed" later in the text), while some event types are only locally predictive (e.g. "he sat down" predict "he stood up," but only nearby). The LSTM is in principle able to learn these sorts of dynamics. Further, since our formulation will require one event to span multiple timesteps across the RNN, handling even short-range dependencies across events requires modeling dependencies across tens of timesteps, so long-range propagation in the network is crucial.

Script Models

We parse a large collection of natural-language documents, extract sequences of events from them, and learn statistical models of these sequences. Critically, we must specify what we mean by an *event*. We adopt a similar event representation to Pichotta and Mooney (2014), modeling an event as a (possibly phrasal) verb with a number of arguments, each of which may have noun information, coreference information, or both. Our proposed LSTM model could straightforwardly handle events with arbitrarily many arguments; however, in order to compare to previously published systems with fixed-arity events, we limit events' arguments to a subject, a direct object, and one prepositional argument (including the preposition, which Pichotta and Mooney (2014) ignore). This amounts to representing events as 5-tuples.

The task we evaluate on, then, is to take as input a sequence of events and output additional events, where an event is a tuple (v, e_s, e_o, e_p, p), where v is a verb lemma, e_s, e_o, and e_p are nominal arguments standing in subject, direct object, and prepositional relations, respectively (about which we may know coreference information, the lemma of the head noun, or both), and p is the preposition relating v and e_p. Any of e_s, e_o, e_p, and p may be *null*, indicating that no word stands in that relation to v (e_p is null if and only if p is null). We will refer to v, e_s, e_o, e_p, and p as *event components*.

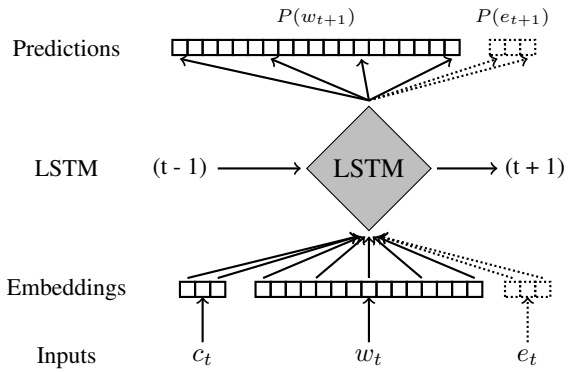

Figure 1: LSTM Script System at timestep t.

LSTM Script Models

We frame script learning as a sequence modeling task, using the standard technique of training a model to sequentially predict the next input. That is, at timestep t, the model is trained to predict the input at timestep $t + 1$. The sequence modeled is the sequence of 5-component events; that is, a sequence of N events has $5N$ timesteps.

We differentiate between two types of script systems based on what the models predict. **Noun models** learn to predict events as verb lemmas, noun lemmas, and prepositions. **Entity models** learn to predict verbs, entity IDs, and prepositions, where an *entity ID* is an integer identifying an argument's entity according to a coreference resolution engine. For example, suppose we observe the two co-occurring events

(*pass, senate, bill*$_1$, $\cdot$, $\cdot$)
(*veto, president, it*$_1$, $\cdot$, $\cdot$)

where $\cdot$ represents a null argument and subscripts indicate entity IDs. An LSTM noun model will be trained to model the sequence (*pass, senate, bill,* $\cdot$, $\cdot$, *veto, president, it,* $\cdot$, $\cdot$), by successively predicting the next element in the sequence (when receiving *pass* as input, it is trained to predict *senate*; in the next timestep it is trained to predict *bill*, and so on). An LSTM entity model will be trained to model (*pass,* 0, 1, $\cdot$, $\cdot$, *veto,* 0, 1, $\cdot$, $\cdot$), where 0 denotes singleton nouns, and 1 is the entity ID for *bill/it*. Previously published statistical event-inference script models are entity models.

We consider four similar model architectures differing in inputs and outputs, depicted in Figure 1 (the inputs and outputs not present in all models have dotted lines). At each timestep t, there are multiple inputs, each of which is a one-hot vector (with one 1 and all other entries 0). First, there is the 1-of-5 input c_t, indicating which component of the event is input at t: verbs will have $c_t = 1$, subject entities $c_t = 2$,

and so on. Next, there is a 1-of-V input w_t, with V the size of the vocabulary, giving the component word at timestep t. Finally, three of the four models have a one-hot e_t input, which gives the entity ID of noun arguments according to a coreference engine. This e_t value has special values for null, singleton entities, and non-entity words (verbs and prepositions). We limit the number of entity IDs to 5,[2] treating all other entities as singletons.

One-hot input vectors are mapped to continuous distributed representations, labeled "Embeddings" in Figure 1. These embeddings are learned jointly with the other model parameters. The embeddings are input to a recurrent LSTM unit, which modifies a latent state vector at each timestep. All models have an output vector from the LSTM, in $\mathbb{R}^V$, which is input to a softmax function, yielding a distribution over predictions for the next w_t value. Additionally, entity models have a second output vector which is input to a softmax predicting the next e_t value. We train all models by minimizing the cross-entropy error at the top softmax layer and backpropagating the error gradient through the network.

We compare four related architectures, which all receive and predict verbs and prepositions but differ in the input and output of entity arguments:

1. **LSTM-noun-noun**, which receives only noun information about arguments and learns to predict argument nouns;
2. **LSTM-ent-ent**, which receives only entity IDs and learns to predict entity IDs;
3. **LSTM-both-noun**, which receives noun and entity IDs and learns to predict nouns;
4. **LSTM-both-ent**, which receives noun and entity IDs and learns to predict entity IDs.

To generate probable event inferences, we perform a five-step beam search over the components (v, e_s, e_o, e_p, p) of events. In steps 2 through 5 of this search, the previous step's output is treated as input. Since the LSTM-both-noun and LSTM-both-ent models require both noun and entity ID information but only predict one of the two, we must generate entity ID information from predicted nouns, and vice versa. When predicting with the LSTM-both-noun model, we call any predicted non-null noun a singleton entity; when predicting with the LSTM-both-ent model, we guess the special Out-Of-Vocabulary token for any predicted non-null entities.

Baseline Models

We compare against four types of baseline script models. The first is the **unigram** model, which infers events by ignoring a document and simply guessing events according to their frequency in the training set. Pichotta and Mooney (2014) showed this baseline to be surprisingly competitive on the Narrative Cloze task. To compare with the different LSTM models presented above, we construct both a **unigram noun** model (with arguments represented by their head noun lemmas) and a **unigram entity** model (with arguments represented by entity IDs).

[2] 98% of training sequences involve five or fewer non-singleton entities.

The other baseline systems we compare to follow previously published systems in making inferences based on event co-occurrence statistics. The simplest is the **all-bigram** system. This system first calculates a closed vocabulary of the most common event types from the training set. Next, event bigram statistics are calculated from the training set, where we follow Jans et al. (2012) and Pichotta and Mooney (2014) in calculating 2-skip bigram counts: when counting co-occurrence, event b is considered to follow event a if b is observed after a with at most two intervening events. Potential inferences at position t in a sequence are scored by maximizing the objective

$$S(a) = \sum_{i=0}^{t-1} \log P(a|s_i)$$

where s_i is the event at position i of the sequence, and $P(b|a)$ is the probability of b following event a according to the 2-skip bigram model.

The third type of baseline system we compare to is the **rewritten all-bigram** model. Similar to the all-bigram model, we calculate 2-skip bigram counts, and events are inferred by maximizing $S(a)$. However, co-occurrences are counted differently: during training, if two events a and b are observed co-occurring, we also "hallucinate" co-occurrences of all events a' and b' such that a' is a with any subset of its entity IDs rewritten, and b' is b with any of its entity IDs rewritten, and shared entities are rewritten consistently. The intuition is that if two events co-occur, then they may also co-occur with different arguments. For more details, see Pichotta and Mooney (2014). This system does not easily admit incorporation of nouns.

The final baseline is the **2D rewritten all-bigram** model, which is the method described by Pichotta and Mooney (2014). This system is similar to the rewritten all-bigram model, with the difference that it maximizes

$$S_2(a) = \sum_{i=0}^{t-1} \log P(a|s_i) + \sum_{i=t+1}^{\ell} \log P(s_i|a)$$

with ℓ the length of the sequence. This model incorporates the probabilities of an event a preceding events after t in the document. This is the best performing published system on the Narrative Cloze evaluation. In order to make these baselines as competitive as possible, if none of the test events s_i being conditioned upon are in the event vocabulary, the models fall back to estimating $S(a)$ with a unigram model.

The baselines do not decompose events into constituent parts, but instead treat entire events as atomic. For example, "Barbarians invade Rome" may be one event type, and "Visigoths sack Rome" another, but the two will be unrelated in the event vocabulary. This requires a very large event vocabulary and will have sparse co-occurrence statistics and poor generalization. On the other hand, in the LSTM script model, *invade*, *sack*, *Barbarian*, and *Visigoth* could individually be in the vocabulary, and each will have a low-dimensional continuous representation in which predictively similar words should be similar. We can therefore predict events never observed in the training set, and never observed

co-occurring with any events in the test document. Further, the LSTM script model allows us to easily input both noun and entity information about arguments.

Evaluation

Experimental Details

We use the Stanford dependency parser (De Marneffe, MacCartney, and Manning 2006) and coreference system (Lee et al. 2013).[3] We lemmatize verbs and directly incorporate negation, a closed set of particles, and XCOMP nodes (clausal complements with external subjects). For example, "Jim didn't dance" produces a `not_dance` event; "Jim took his shirt off" a `take_off` event; and "Jim forgot to take it" a `forget_to_take` event. Passive constructions are normalized to be identical to their active counterparts. We represent noun arguments by their head lemmas.

For our corpus, we use English Language Wikipedia,[4] breaking articles into paragraphs. Our training set was approximately 8.9 million event sequences, our validation set was approximately 89,000 event sequences, and our test set was 2,000 events from 411 sequences, such that no test-set article is in the training or validation set. We add a <s> beginning-of-sequence pseudo-event and a </s> end-of-sequence pseudo-event to every sequence. The event component vocabulary comprises the 2,000 most common verbs, the 8,000 most common nouns, and the top 50 prepositions; all other words are replaced with an Out-Of-Vocabulary (OOV) token. For the unigram and bigram event vocabulary, we select the 10,000 most common events (with either nouns or entity IDs, depending on the system). We apply add-one Laplace smoothing to bigram co-occurrence counts.

We use the implementation of LSTM provided by the Caffe library (Jia et al. 2014). We train using batch stochastic gradient descent with momentum with a batch size of 20. Since RNNs are quite sensitive to hyperparameter values (Sutskever et al. 2013), we measured validation set performance in different regions of hyperparameter space, ultimately selecting learning rate $\eta = 0.1$, momentum parameter $\mu = 0.98$, LSTM vector length of 1,000, and a Normal $\mathcal{N}(0, 0.1)$ distribution for random initialization (biases are initialized to 0). Event component embeddings have dimension 300. We use ℓ_2 regularization and Dropout (Hinton et al. 2012) with dropout probability 0.5. We clip gradient updates at 10 to prevent exploding gradients (Pascanu, Mikolov, and Bengio 2013) We damp η by 0.9 every 100,000 iterations. We train for 750,000 batch updates, which took between 50 and 60 hours. We use a beam width of 50 in all beam searches.

Evaluating Held-Out Event Inferences

We first evaluate using the Narrative Cloze evaluation (Chambers and Jurafsky 2008), which tests a system's ability to reconstruct held-out events from documents. This evaluation is fully automated and does not require manually labeled evaluation data. We use four metrics. Primarily, we evaluate with **Recall at 25 ("R25")**, which is the percentage of held-out events that appear in the top 25 guesses a system makes for that event; recall-at-k was also used in Jans et al. (2012) and Pichotta and Mooney (2014). Since an inference is counted as "correct" only if all event components match the held-out exactly, this is a very challenging task in the multi-argument setting, particularly when predicting nouns. We also calculate **Verb recall at 25 ("R25-V")**, which is recall at 25, but counting an inference as correct if its verb matches the held-out event's verb (ignoring arguments), and **4-Tuple recall at 25 ("R25-4")**, which is recall at 25 ignoring prepositions. This allows us to compare directly to the method of Pichotta and Mooney (2014), whose events contain no prepositions. We evaluate LSTM systems by predicting 5-tuples and discarding prepositions, and evaluate baseline systems by directly modeling (v, e_s, e_o, e_p) 4-tuples.

Finally, motivated by the stringency of the R25 metric, especially in the noun setting, we evaluate using **Accuracy with Partial Credit ("Acc")**, in which we compute a system's single most confident inference and calculate, for every component of the held-out event, a similarity score between that component and the respective inferred component. This relaxes the requirement that inferred events match exactly, giving partial credit for similar components. Partial credit is computed using WUP similarity (Wu and Palmer 1994), based on distance in the WordNet hierarchy (Fellbaum 1998). We assign a similarity score by taking the maximum WUP scores over all Synset pairs (with appropriate parts-of-speech). Accuracy is average WUP score across event components (ignoring OOVs and nulls in the held-out event). This will be between 0 and 1. We use the NLTK implementation of WUP (Bird et al., 2009).

Table 1 gives results on the Narrative Cloze evaluation. The LSTM-both-ent system demonstrates a **50.0%** relative improvement (5.7% absolute improvement) over the current best-published system (2D rewritten all-bigram, evaluated using 4-Tuple event recall at 25). Note that the simpler all-bigram system outperforms the rewritten versions. This is probably because there is information encoded in the entity IDs (the relative ordering of entities, and which entities are singletons) that is lost during rewriting. Note also that, on this corpus, the 2D rewritten system, which makes predictions based on subsequent events in addition to previous events, does marginally worse than the system using only previous events. We hypothesize this is because subsequent events are less predictive than previous events on this corpus, and are comparatively overweighted.

Compared to the strongest baselines, the best-performing entity system achieves a **22.6%** relative improvement on R25, an **18.4%** relative improvement on verb-only R25, and an **8.8%** relative improvement on accuracy with partial credit. The best-performing noun system achieves a **64.9%** relative improvement on R25, a **33.9%** relative improvement on verb-only R25, and an **18.2%** relative improvement on accuracy with partial credit. LSTM-both-ent is the best entity model, and LSTM-both-noun is the best noun model; that is, the best performing system in both cases is the one which is given both noun and entity information.

[3] We use version 3.3.1 of the Stanford CoreNLP system.

[4] `http://en.wikipedia.org/`, dump from Jan 2, 2014.

	Entities				Nouns			
System	**R25**	**R25-V**	**R25-4**	**Acc.**	**R25**	**R25-V**	**R25-4**	**Acc.**
Unigram	0.101	0.192	0.109	0.402	0.025	0.202	0.024	0.183
All-Bigram	0.124	0.256	0.140	0.420	0.037	0.224	0.039	0.220
Rewrite Bigram	0.110	0.205	0.125	0.421	-	-	-	-
2D Rewrite Bigram	0.104	0.192	0.114	0.416	-	-	-	-
LSTM-ent-ent	0.145	0.279	0.160	0.450	-	-	-	-
LSTM-both-ent	**0.152**	**0.303**	**0.171**	**0.458**	-	-	-	-
LSTM-noun-noun	-	-	-	-	0.054	0.298	0.057	0.256
LSTM-both-noun	-	-	-	-	**0.061**	**0.300**	**0.062**	**0.260**

Table 1: Narrative Cloze results on entity and noun models, with four metrics (higher scores are better).

Evaluating Novel Event Inferences

The low magnitude of the Narrative Cloze scores reflects the task's difficulty. The evaluation has a number of intuitive shortcomings: by their very nature, most obviously inferred facts are not explicitly stated, and so are not captured. Also, Cloze scores on individual held-out events are not easily interpretable. Motivated by these concerns, we evaluate inferences by eliciting human judgments via Amazon Mechanical Turk. Given a text snippet, annotators are asked to rate, on a 5-point Likert scale, the likelihood of inferences, with 5 signifying "Very Likely" and 1 "Very Unlikely/Irrelevant" (uninterpretable events are to be marked "Nonsense"). This provides interpretable scores, and allows us to directly compare entity- and noun-predicting models, which is not straightforward using the Narrative Cloze.

We present annotators with a snippet of text and 5 phrases, 4 of which are automatic script inferences based on the events in the snippet, and one of which is a randomly selected event from the 10,000 most frequent events ("Random"). We transform relational events to English phrases using an LSTM model trained to predict, from extracted event tuples, the original text from which the event was extracted. This network uses a hidden state vector of length 1,000 and a vocabulary of 100k tokens. We elicit three judgments for each inference, treating "nonsense" judgments as 0 scores.

We asked annotators to judge each system's most confident inference not involving one of the ten most frequent verbs in the corpus.[5] We evaluate two noun-predicting systems: LSTM-both-noun and All-bigram-noun, which were the best-performing LSTM and Bigram systems on the Narrative Cloze; we also collect judgments for two entity systems, LSTM-both-ent and All-bigram-ent. We collect judgments on inferences from 100 snippets, each of which is the smallest set of initial sentences from a different paragraph in the test set such that the text contains at least two events.

The "All" column in Table 2 gives average ratings for each system. The "Filtered" column gives the results after removing annotations from annotators whose average "Random" score is higher than 1.0 (this may be viewed as a quality-control procedure). The LSTM-both-noun system, which predicts verbs and nouns, significantly outperforms all other systems, both with and without filtering ($p < 0.05$, Wilcoxon-Pratt signed-rank test). Incorporating nouns into LSTM models improves inferences; on the other hand, bigram models, which do not decompose events into constituent components, perform worse when directly incorporating nouns, as this increases event co-occurrence sparsity.

[5] *have, make, use, include, know, take, play, call, see, give*.

System	**All**	**Filtered**
Random	2.00	0.87
All-Bigram Ent	2.87	2.87
All-Bigram Noun	2.47	2.21
LSTM-both-ent	3.03	3.08
LSTM-both-noun	**3.31**	**3.67**

Table 2: Crowdsourced results (scores range from 0 to 5).

Qualitative Analysis

Figure 2 (Top) shows, for two short two-event test sequences, the top 3 inferences the LSTM-both-noun system makes at each position (the inferences following an event are the system's top predictions of immediately subsequent events). Subscripts are entity IDs (singleton entities are unsubscripted). We do not display bigram inferences, because in these examples they are exactly the most-common unigram events, as no observed events are in the event vocabulary. These examples clearly illustrate the importance of incorporating argument noun information: for example, without nouns, *(obtain, OOV$_1$, phd, dissertation, with)* would be represented as, roughly, "someone obtained something with something," from which few reasonable inferences can be made. Note that since the learning objective does not directly encourage diversity of inferences, the LSTM makes a number of roughly synonymous inferences.

To get further intuitions for what these models learn, we can seed a model with a <s> beginning-of-sequence event and generate events by probabilistically sampling from its output predictions until it generates </s> ("ask it to generate a story"). That is, the first event component (a verb) is sampled from the model's learned distribution of first components, the hidden state is updated with this sample, the next component is sampled from the model's predictions, and so on, until a </s> is sampled. Figure 2 (Bottom) gives three probabilistically generated sequences from the LSTM-noun-noun model. These sequences, generated totally from scratch one component at a time, are reasonably coherent, and exhibit clear thematic dependencies across events.

Sample Events Inferred from Test Data

Sequence 1 (two events):		
Event 1:	**(obtain, OOV$_1$, phd, dissertation, with)**	**___ obtained a PhD with a dissertation**
Inference 1:	(study, he, ., university, at)	He studied at a university
Inference 2:	(study, OOV, ., university, at)	___ studied at a university
Inference 3:	(study, he, ., OOV, at)	He studied at ___
Event 2:	**(graduate, OOV$_1$, ., university, at)**	**___ graduated at a university**
Inference 1:	(move, he, ., OOV, to)	He moved to ___
Inference 2:	(move, OOV, ., OOV, to)	___ moved to ___
Inference 3:	(return, he, ., OOV, to)	He returned to ___
Sequence 2 (two events):		
Event 1	**(destroy, ., airport$_1$, 1945, in)**	**The airport was destroyed in 1945.**
Inference 1:	(destroy, ., airport, ., .)	The airport was destroyed
Inference 2:	(rebuild, ., airport, ., .)	The airport was rebuilt
Inference 3:	(build, ., airport, ., .)	The airport was built
Event 2	**(open, airport$_1$, ., 1940, in)**	**The airport opened in 1940**
Inference 1:	(rename, ., airport, ., .)	The airport was renamed
Inference 2:	(know, ., ., airport, as)	. . . known as ___ airport
Inference 3:	(use, ., airport, ., .)	The airport was used

Probabilistically Generated Event Sequences

((pass, route, creek, north, in);	The route passes the creek in the North
(traverse, it, river, south, to))	It traverses the river to the South
((issue, ., recommendation, government, from);	A recommendation was issued from the government
(guarantee, ., regulation, ., .);	Regulations were guaranteed
(administer, agency, program, ., .);	The Agency administered the program
(post, ., correction, website, through);	A correction was posted through a website
(ensure, standard, ., ., .);	Standards were ensured
(assess, ., transparency, ., .))	Transparency was assessed.
((establish, ., ., citizen, by)	Established by citizens, . . .
(end, ., liberation, ., .)	. . . the liberation was ended
(kill, ., man, ., .)	A man was killed
(rebuild, ., camp, initiative, on)	The camp was rebuilt on an initiative
(capture, squad, villager, ., .)	A squad captured a villager . . .
(give, inhabitant, group, ., .))	. . . [which] the inhabitants had given the group

Figure 2: Sample Narrative Cloze inferences (Top), and some probabilistically generated event sequences (Bottom). The right column gives possible English descriptions of the structured events on the left.

Related Work

Scripts in AI date back to the 1970s (Schank and Abelson 1977). Mooney and DeJong (1985) give an early knowledge-based method for learning scripts from a single document. Miikkulainen (1993) proposes a Neural Network system which stores events in episodic memory and is capable of answering simple questions.

There has been a fair amount of recent work on learning models of events from large corpora. Some systems are focused on event inference (Chambers and Jurafsky 2008; Bejan 2008; Jans et al. 2012; Pichotta and Mooney 2014; Rudinger et al. 2015); others focus on learning structured collections of events (Chambers 2013; Cheung, Poon, and Vanderwende 2013; Balasubramanian et al. 2013; Bamman and Smith 2014; Nguyen et al. 2015); others focus on story generation (McIntyre and Lapata 2009; 2010). There is also a body of work on building high-precision models of real-world situations from smaller corpora of event sequences (Regneri, Koller, and Pinkal 2010; Li et al. 2012; Frermann, Titov, and Pinkal 2014; Orr et al. 2014).

There have been a number of recent papers successfully applying neural nets to tasks above the sentence level. Li, Li, and Hovy (2014) and Ji and Eisenstein (2015) use RNNs for discourse parsing. Li and Hovy (2014) and Modi and Titov (2014) present neural models of event ordering. Kalchbrenner and Blunsom (2013) use RNNs to classify speech acts. Weston, Chopra, and Bordes (2015) describe a model with a long-term memory component, which they use to answer questions about short generated stories.

Conclusion

We have presented an LSTM-based statistical script model capable of modeling and predicting either noun or coreference information about event arguments. We compared to a number of baselines, including the previous best-published system, on the tasks of inferring both held-out and novel events, demonstrating substantial improvements.

Acknowledgments

Thanks to Saro Meguerdichian, Matt Hausknecht, Amelia Harrison, and the UT NLP group for discussions and feedback. This research was supported in part by the DARPA DEFT program under AFRL grant FA8750-13-2-0026.

References

Balasubramanian, N.; Soderland, S.; Mausam; and Etzioni, O. 2013. Generating coherent event schemas at scale. In *EMNLP*.

Bamman, D., and Smith, N. 2014. Unsupervised discovery of biographical structure from text. *TACL*.

Bejan, C. A. 2008. Unsupervised discovery of event scenarios from texts. In *FLAIRS*.

Bird, S.; Klein, E.; and Loper, E. 2009. *Natural Language Processing with Python*.

Chambers, N., and Jurafsky, D. 2008. Unsupervised learning of narrative event chains. In *ACL*.

Chambers, N., and Jurafsky, D. 2009. Unsupervised learning of narrative schemas and their participants. In *ACL*.

Chambers, N. 2013. Event schema induction with a probabilistic entity-driven model. In *EMNLP*.

Cheung, J. C. K.; Poon, H.; and Vanderwende, L. 2013. Probabilistic frame induction. In *NAACL*.

De Marneffe, M.-C.; MacCartney, B.; and Manning, C. D. 2006. Generating typed dependency parses from phrase structure parses. In *LREC*, volume 6.

Donahue, J.; Hendricks, L. A.; Guadarrama, S.; Rohrbach, M.; Venugopalan, S.; Saenko, K.; and Darrell, T. 2015. Long-term recurrent convolutional networks for visual recognition and description. In *CVPR*.

Fellbaum, C. 1998. *WordNet: An Electronic Lexical Database*.

Frermann, L.; Titov, I.; and Pinkal, M. 2014. A hierarchical Bayesian model for unsupervised induction of script knowledge. In *EACL*.

Graves, A.; Mohamed, A.-r.; and Hinton, G. 2013. Speech recognition with deep recurrent neural networks. In *ICASSP*.

Hinton, G. E.; Srivastava, N.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. R. 2012. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv*.

Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural Computation* 9(8).

Jans, B.; Bethard, S.; Vulić, I.; and Moens, M. F. 2012. Skip n-grams and ranking functions for predicting script events. In *EACL*.

Ji, Y., and Eisenstein, J. 2015. One vector is not enough: Entity-augmented distributional semantics for discourse relations. *TACL*.

Jia, Y.; Shelhamer, E.; Donahue, J.; Karayev, S.; Long, J.; Girshick, R.; Guadarrama, S.; and Darrell, T. 2014. Caffe: Convolutional architecture for fast feature embedding. *arXiv*.

Kalchbrenner, N., and Blunsom, P. 2013. Recurrent convolutional neural networks for discourse compositionality. In *CVSC*.

Lee, H.; Chang, A.; Peirsman, Y.; Chambers, N.; Surdeanu, M.; and Jurafsky, D. 2013. Deterministic coreference resolution based on entity-centric, precision-ranked rules. *Computational Linguistics* 39(4).

Li, J., and Hovy, E. 2014. A model of coherence based on distributed sentence representation. In *EMNLP*.

Li, B.; Lee-Urban, S.; Appling, D. S.; and Riedl, M. O. 2012. Crowdsourcing narrative intelligence. *Advances in Cognitive Systems* 2.

Li, J.; Li, R.; and Hovy, E. 2014. Recursive deep models for discourse parsing. In *EMNLP*.

McIntyre, N., and Lapata, M. 2009. Learning to tell tales: A data-driven approach to story generation. In *ACL*.

McIntyre, N., and Lapata, M. 2010. Plot induction and evolutionary search for story generation. In *ACL*.

Miikkulainen, R. 1993. *Subsymbolic Natural Language Processing: An Integrated Model of Scripts, Lexicon, and Memory*.

Modi, A., and Titov, I. 2014. Inducing neural models of script knowledge. In *CoNLL*.

Mooney, R. J., and DeJong, G. F. 1985. Learning schemata for natural language processing. In *IJCAI*.

Nguyen, K.-H.; Tannier, X.; Ferret, O.; and Besançon, R. 2015. Generative event schema induction with entity disambiguation. In *ACL*.

Orr, J. W.; Tadepalli, P.; Doppa, J. R.; Fern, X.; and Dietterich, T. G. 2014. Learning scripts as Hidden Markov Models. In *AAAI*.

Pascanu, R.; Mikolov, T.; and Bengio, Y. 2013. On the difficulty of training recurrent neural networks. In *ICML*.

Pichotta, K., and Mooney, R. J. 2014. Statistical script learning with multi-argument events. In *EACL*.

Regneri, M.; Koller, A.; and Pinkal, M. 2010. Learning script knowledge with web experiments. In *ACL*.

Rudinger, R.; Rastogi, P.; Ferraro, F.; and Van Durme, B. 2015. Script induction as language modeling. In *EMNLP*.

Schank, R. C., and Abelson, R. P. 1977. *Scripts, Plans, Goals and Understanding: An Inquiry into Human Knowledge Structures*.

Sundermeyer, M.; Schlüter, R.; and Ney, H. 2012. LSTM neural networks for language modeling. In *INTERSPEECH*.

Sutskever, I.; Martens, J.; Dahl, G.; and Hinton, G. 2013. On the importance of initialization and momentum in deep learning. In *ICML*.

Sutskever, I.; Vinyals, O.; and Le, Q. V. 2014. Sequence to sequence learning with neural networks. In *NIPS*.

Venugopalan, S.; Xu, H.; Donahue, J.; Rohrbach, M.; Mooney, R.; and Saenko, K. 2015. Translating videos to natural language using deep recurrent neural networks. In *NAACL*.

Weston, J.; Chopra, S.; and Bordes, A. 2015. Memory networks. In *ICLR*.

Wu, Z., and Palmer, M. 1994. Verbs semantics and lexical selection. In *ACL*.

Zaremba, W., and Sutskever, I. 2014. Learning to execute. *arXiv*.