Zero-Shot Script Parsing

Anonymous ACL submission

Abstract

Script knowledge (Schank and Abelson, 1977) proved useful to a variety of NLP tasks. However, existing resources only covering a small number of activities, limiting its practical usefulness. In this work, we propose a zero-shot learning approach to **script parsing**, the task of tagging texts with pre-defined, scenario-specific event and participant types, which makes it possible to acquire script knowledge without domain-specific annotations. We (1) learn representations of potential event and participant mentions by promoting cluster consistency according to the annotated data; (2) perform clustering on the event / participant candidates from unannotated texts that belongs to an unseen scenario. We further exploit dependency and coreference information. The model achieves 68.1/74.4 average F1 for event / participant parsing, respectively, outperforming a previous CRF model that has access to domain-specific supervision.

1 Introduction

Script knowledge is a type of commonsense knowledge that captures how people conduct everyday activities (Schank and Abelson, 1977). It expresses that in a certain **scenario**, **participants** tend to act out **events** in a certain order; an example from the scenario FIXING A FLAT TIRE is shown in Fig. 1. Humans use script knowledge to fill in events that are not explicitly mentioned in a text, and script knowledge is useful for many downstream NLP applications, including referent prediction (Ahrendt and Demberg, 2016; Modi et al., 2017), discourse sense classification (Lee et al., 2020), and story generation (Zhai et al., 2019, 2020).

A key challenge in dealing with script knowledge is coverage: it is costly and time-consuming to spell out the prototypical events and participants of a scenario and how they can be expressed in language. Crowd-sourced script resources (Regneri et al., 2010; Modi et al., 2016) address this issue by annotating stories with script events and participants (cf. Fig. 1). **Script parsers**, which predict these event and participant labels given a text, can achieve high accuracies on scenarios that were seen in training (Ostermann et al., 2017; Zhai et al., 2021). But the script resources are still limited in coverage (for instance, the InScript corpus of Modi et al. (2016) covers ten scenarios), which limits the practical usefulness of script parsers and thus the practical usefulness of script knowledge for downstream tasks in general.

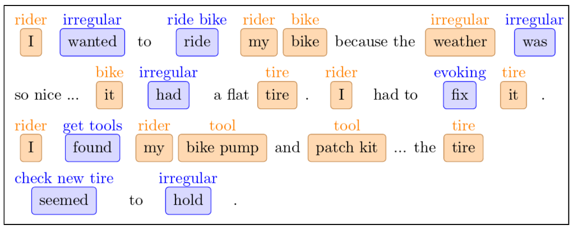


Figure 1: A story about FIXING A FLAT TIRE from InScript. Script parsing identifies events and participants from texts. The picture is taken from Zhai et al. (2021).

In this paper, we tackle the task of **zero-shot script parsing**: we present the first system which accurately performs script parsing on scenarios that were not seen at training time. For instance, given training data that talks about taking a bath and going to a restaurant, the parser labels events and participants in the FIXING A FLAT TIRE story of Fig. 1. This offers a way of overcoming the coverage limitations of script knowledge, by generalizing from the training scenarios to arbitrary other ones. Our method learns to extract script-specific representations from general-purpose pretrained word embeddings, and then uses agglomerative clustering at inference time to group together natural-language phrases that refer to the same event or participant of the unseen script.

Our model achieves a micro-F1 score on zero-shot event labeling of up to 68.1 and a micro-F1 on participant labeling of up to 74.4, on par with the supervised model of Ostermann et al. (2017) that assumes training data for the same scenario. We find that our method yields script graphs with reasonable event clusters that are temporally ordered in a reasonable way; the majority of errors on event labeling are due to issues with the granularity of events. We also find in probing tasks that our model learns to amplify information about sentence ordering from the pretrained embeddings, while suppressing low-level information about morphology and syntax, which are less relevant for the script parsing task.

2 Related work

Scripts were introduced as an approach to capturing commonsense knowledge in AI by Schank and Abelson (1977); see also Barr and Feigenbaum (1981). Much research in NLP has simplified the learning of script knowledge to identifying "event chains" in narrative text (Chambers and Jurafsky, 2008, 2009). Event chains represent typical sequences of events, each represented by one verb, and can be learned from large corpora. Other work has followed in this tradition (Jans et al., 2012; Modi and Titov, 2014; Pichotta and Mooney, 2014; Rudinger et al., 2015).

In this paper, we instead build upon work by Regneri et al. (2010, 2011), who explicitly capture script knowledge about a given scenario in a *temporal script graph* (see Fig. 4). A TSG specifies the abstract events and participants that make up a script with their temporal ordering; each of these events and participants can be expressed in language in many different ways. Regneri et al. learned script graphs by crowdsourcing. We instead rely on manually script-annotated corpora (Modi et al., 2016; Wanzare et al., 2016).

With scenario-specific supervision, script parsing can be performed accurately. Ostermann et al. (2017) developed a linear CRF model to perform script parsing as a sequence labelling task. Zhai et al. (2021) developed a hierarchical model for supervised script parsing, making use of pre-trained contextualized word embeddings. The model learns patterns at the level and the narrative level with respective sequence models. These existing approaches are limited to scenarios for which training data is available, whereas our work focuses on unseen scenarios.

Zero-shot learning is a family of methods that establishes a classifier for **unseen** classes, based on labelled data from **seen** classes. One common approach is to learn a latent representation space that all instances embed into, thus the knowledge of the source domain, encoded in the labelled training instances, could be transferred to the target domain. It tackles data scarcity in various situations, such as machine translation for low-resource languages (e.g. Pham et al., 2019; Zhang et al., 2020; Johnson et al., 2017), generation (Duan et al., 2019; Philip et al., 2020), text classification (see, e.g. Yin et al., 2019) and question answering(e.g. Banerjee and Baral, 2020).

3 Data and task

We work with **InScript** (Modi et al., 2016), a crowdsourced corpus of around 100 stories about each of 10 scenarios (see Fig. 1 for an example). The authors were asked to write a story about a given scenario (such as GOING GROCERY SHOPPING) "as if to a child", step by step. InScript was then hand-annotated with event and participant classes; it also contains coreference and dependency annotations.

In this paper, we consider the task of predicting event and participant annotations for a scenario that was not seen in training. Thus, our model must learn to group verbs and noun phrases from an unseen scenario into abstract events and participants, without knowing what the gold events and participants are. We split InScript into eight training, one validation, and one test scenario. During inference, the model takes the unannotated stories of the test scenario as input and must label them with events and participants that are consistent with the gold annotations.

Following Ostermann et al. (2017), we distinguish between (1) events that are 'related to the scenario', or commonly seen in a typical instantiation of the scenario, which we call **regular events**, and (2) the ones that take place in the course of a specific story, but are not directly related to the scenario, which we call **irregular events**. For example, in Figure 1, 'I found my bike pump' describes the regular event 'get tools', whereas *the weather **was** nice* is irregular. We collapse all the subclasses of irregular events, UNREL, RELNSCR, OTHER and UNKNOWN, into a single irregular event class for each scenario. 12,902 (33.5%) event instances in InScript are regular. We also distinguish **regu-**

lar participants from **irregular participants** in a similar manner: participants like 'rain' in Fig. 1 are considered irregular to the FIXING A FLAT TIRE scenario, as they are not directly relevant to the scenario per se. Irregular participant instances take a smaller proportion of 19.6%.

4 Method

The basic idea of our zero-shot script parser is as follows. We will learn a transformation φ which maps pretrained general-purpose word embeddings into a representation space that is suitable for script parsing. Identifying verb tokens as *candidates* for event descriptions and noun and pronoun tokens as candidates for participant descriptions, we will train φ such that candidates that describe the same event or participant are close together in the representation space, whereas candidates for different events and participants are distant. To parse a text from an unseen scenario, we will apply φ to the word embeddings of all candidates and perform clustering to group them into events and participants.

4.1 Regular candidate identification

Throughout the paper, we will focus on *regular* candidates, because irregular candidates are a diverse group without a tight semantic connection to the scenario, and may not cluster easily in the representation space. We ignore irregular candidates in training. During inference, we evaluate against the original gold standard.

We train a classifier to distinguish regular and irregular candidates so the latter could be excluded from training. We use the same architecture as in the supervised script parser of Zhai et al. (2021), but trained only to distinguish regular candidates from irregular candidates. We obtain training data for this task by grouping the original labels into one of REGULAR_EVENT, IRREGULAR_EVENT, REGULAR_PARTICIPANT and IRREGULAR_PARTICIPANT. The model is trained on the 8 training scenarios and validated on the validation scenario. Finally, we perform inference on the test scenario. The classifier achieves on average 85 points F1-score.

4.2 Training

We will now describe how to learn φ. For any given text that we want to parse, we will run XLNet (Yang et al., 2019) to obtain contextualized word embeddings $f(c)$ for each event and participant candidate c. We will then train φ to minimize distances within the same event and participant class and maximize them between different ones (§4.2.1); the general framework is illustrated in Fig. 2. We will then describe several extensions to the loss function (§4.2.2–§4.2.3) and then discuss replacing XLNet embeddings with more specialized word embeddings (§4.2.4).[1].

4.2.1 Learning script-specific representations

Let $\mathcal{C}$ be the set of all event candidates or the set of all participant candidates in a text, and let $\pi(\mathcal{C})$ be a partition of $\mathcal{C}$ which clusters candidates into equivalence classes; at training time, each class contains the candidates that are labeled with the same event or participants. We define $\pi(c)$ as the element of the partition to which the candidate c belongs. Given a pre-trained embedding function f and the transformation φ_θ that we want to learn, we consider the average distance between instances belonging to different clusters:

$$d_{ext}(\pi(\mathcal{C});\theta) = \operatorname*{\mathit{mean}}_{\substack{c,c' \in \mathcal{C}:\\ \pi(c) \neq \pi(c')}} d(\varphi_\theta(f(c), \varphi_\theta(f(c'))$$

We would like to push the embeddings of two candidates from different classes away if they are too close to each other. We do so by maximizing the *external consistency* of the partition π:

$$\gamma_{ext}(\pi(\mathcal{C});\theta) = \operatorname*{\mathit{mean}}_{\substack{c,c' : \pi(c) \neq \pi(c'),\\ d(\varphi_\theta(f(c), \varphi_\theta(f(c'))\\ < \sigma_1 d_{ext}(\pi(\mathcal{C});\theta)}} d(\varphi_\theta(f(c), \varphi_\theta(f(c'))$$

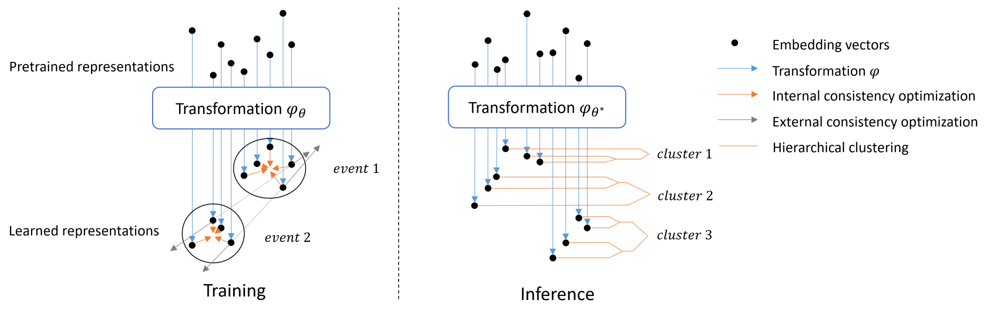
$\sigma_1 \in (0,1)$ is a threshold that quantifies being 'too close'. This definition captures the intuition that φ should map candidates from different classes to dissimilar vectors. Likewise, consider the average distances between embeddings of candidates from same classes:

$$d_{int}(\pi(\mathcal{C});\theta) = \operatorname*{\mathit{mean}}_{c,c':\pi(c)=\pi(c')} d(\varphi_\theta(f(c), \varphi_\theta(f(c'))$$

We would like to pull the embeddings of two candidates from the same class towards each other if they are too far away. In a similar spirit, we maximize the *internal consistency* of π:

$$\gamma_{int}(\pi(\mathcal{C});\theta) = 1 - \operatorname*{\mathit{mean}}_{\substack{c,c' : \pi(c) = \pi(c'),\\ d(\varphi_\theta(f(c), \varphi_\theta(f(c'))\\ > \sigma_2 d_{int}(\pi;\theta)}} d(\varphi_\theta(f(c), \varphi_\theta(f(c')) \tag{1}$$

[1]See the appendix for more implementation details; we will release our code upon acceptance.



Pretrained representations
Transformation φ_θ
event 1
event 2
Learned representations
Training
Transformation φ_{θ^*}
cluster 1
cluster 2
cluster 3
Inference
Embedding vectors
Transformation φ
Internal consistency optimization
External consistency optimization
Hierarchical clustering

Figure 2: The overall framework. We learn a representation from annotated corpus and apply it to unannotated texts. The coreference and dependency terms are not visualized.

(a) There is a **bus** stop down the street from my house . If you take **it** going south , **it** leads to the city...

(b) ...$I_{passenger}$ **fed** my $coins_{money}$ into the slot where you put your money...
...$I_{passenger}$ boarded the bus and **paid** for my ride with my $change_{money}$...

(c) ...the bus $arrived_{bus\_stops}$ at the **bus stop** closest to the beach...
...I would need the bus to $stop_{bus\_stops}$ next to **the hospital**...

Figure 3: Illustrations of the refined consistency measures.

We write $d(\cdot,\cdot)$ for the distance function in the representation space, with values in $[0,1]$. Empirically, the following variant of cosine distance worked well:

$$\sqrt{\frac{1}{2}-\frac{1}{2}cos(\angle(v,w))}$$

Here $\angle(v,w)\in[0,\pi]$ denotes the angle between v,w.

We obtain an overall consistency measure γ, which we maximize in training; λ_i is a hyper-parameter that balances the terms.

$$\gamma(\pi(\mathcal{C});\theta)=\gamma_{ext}(\pi(\mathcal{C});\theta)+\lambda_i\gamma_{int}(\pi(\mathcal{C});\theta)$$

4.2.2 Coreference

We can now further refine this baseline consistency model with script-specific knowledge. First, within a text, noun phrases that refer to the same entity form a **coreference chain**: for example, all mentions of the bus in the scenario TAKING A BUS (Fig. 3a). As they refer to the same entity, these noun phrases should belong to the same participant cluster and thus have similar representations.

We capture this intuition as follows. Let $\eta(\mathcal{C}^c)$ be the set of all coreference chains on participant candidates. Like π above, $\eta(\mathcal{C}^c)$ also specifies an equivalence relation, in that two candidates are in the same class iff they are in the same coreference chain. We can thus formulate a coreference-based consistency measure as

$$\beta(\theta):=\gamma_{int}(\eta(\mathcal{C}^c);\theta)$$

Note that this is a soft constraint; coreferent entities are rewarded for being in the same class, not forced into them. This increases robustness against noise in the coreference annotations.

4.2.3 Event-participant dependencies

Second, events and participants in a script are tightly linked: if two verbs have arguments from the same participant class, they tend to describe the same event (Fig. 3b); and if two noun phrases are arguments of the same event, they tend to describe the same participant (Fig. 3c).

Let c_p be the set of event candidates that have participant p as an argument; we would like to encourage φ to map the elements of c_p to similar representations. Let $\xi(\mathcal{C}^d_e)$ be the set of all c_p. Analogously, let $\xi(\mathcal{C}^d_p)$ be the set of participant candidate sets that depend on the same events. We can formulate a dependency-based consistency measure as

$$\alpha(\theta)=\gamma_{int}(\xi(\mathcal{C}^e);\theta)+\gamma_{int}(\xi(\mathcal{C}^p);\theta)$$

The final training objective, with hyperparameters λ_c, λ_d and cluster assignment $\pi^*(\mathcal{C}^e)$ of event candidates and $\pi^*(\mathcal{C}^p)$ of participant candidates in InScript, is

$$\begin{aligned}\theta^*=\underset{\theta}{argmax}[\gamma(\pi^*(\mathcal{C}^e);\theta)+\lambda_p\gamma(\pi^*(\mathcal{C}^p);\theta)\\+\lambda_c\beta(\theta)+\lambda_d\alpha(\theta)]\quad(2)\end{aligned}$$

4.2.4 Specialized word embeddings

We further investigated whether our zero-shot approach can benefit by using more specialized word embeddings as input instead of the general-purpose XLNet embeddings. We thus replaced f with the representations from the pre-final layer of the supervised script parser of Zhai et al. (2021). These representations are also based on XLNet, but then trained to predict InScript events and participants on known scenarios.

We deviate from Zhai et al.'s training setup in two ways. **(1)** Data. In order not to neutralize the zero-shot setting, we train the model from Zhai et al. on the 9 scenario we reserved for training and validation, whereas keep the test scenario unseen. **(2)** The parser is trained on regular event/participants only, to be consistent with the clustering settings. Its performance is at 95 points F1-score on its validation set.

4.3 Inference

4.3.1 Clustering

At inference time, we first determine the event and participant candidates by taking the nouns, pronouns and verbs, and classify them for regularity. We then acquire a representation $\varphi_{\theta^*}(f(c))$ for each candidate c and group them into classes by clustering (cf. Fig. 2).

We use agglomerative clustering, a bottom-up hierarchical clustering algorithm that iteratively merges the most similar pair of clusters. It terminates when either the number of clusters decreases to a pre-defined quantity or the minimum dis-similarity between the current clusters goes beyond a predefined threshold. As the number of event and participant classes vary across scenarios, we do not fix the number of cluster, but instead define a dissimilarity threshold estimated from the training scenarios. If the number of clusters resulting from this process was too extreme (>30 or < 10), we reran the process to yield 20 clusters.

4.3.2 Protagonists

As one final optimization, we give special treatment to the **protagonist** of each scenario – for example, the passenger in TAKING A TRAIN or the customer in GROCERY SHOPPING. The protagonist is the most frequent participant in all scenarios and always makes the largest class of participant candidates. We thus identify it by following the longest coreference chain. This simple heuristic yields an F-score of 98 at inference time for the protagonist class. We thus ignore protagonists in training.

5 Evaluation

We evaluate our method with 10-fold cross-validation on InScript by alternating the selection of validation and test scenarios. Note that the texts in the validation and test data are always from scenarios that were unseen in training.

5.1 Metric

Given a cluster assignment, what we are interested in is how well the predicted clusters align with gold classes. We seek to establish a 'best' assignment of the clusters to the gold classes, with which we can evaluate the 'accuracy' of the clustering results as if it were a classification task. One approach is to find the assignment that maximizes this accuracy. This is a *linear assignment* problem, which is solved in cubic time by the Hungarian algorithm (see, e.g. Kuhn, 1955), thus tractable given the scale of our problem. We call the F1 score evaluated according to this optimal assignment **Hungarian F1**, and use it as our main evaluation metric. This metric allows us to compare the results of the clustering-based parsers to that of the classification-based parsers.

5.2 Baselines

We compare the results of our zero-shot parser to a number of baselines. First, we compare against the supervised script parsers of Zhai et al. (2021) and Ostermann et al. (2017) (retrained on the train-test split of Zhai et al.). These parsers are evaluated on a subset of InScript that contains texts from the same scenarios as in the training set, thus the numbers are not directly comparable to ours.

Second, we compare against a baseline where we cluster event and participant candidates at inference time based on the bare XLNet embeddings, rather than the ones that were transformed by our learned φ_{θ^*}. Finally, in addition to our *full* model, as specified by 2 with the specialized embeddings of §4.2.4, we also present results for ablated versions without the extensions regarding event-participant dependencies (*dep*), coreference (*coref*), and specialized embeddings (*specialized*).

For each of the clustering-based methods, we report two results: one where we assume gold information about whether an event or participant candidate is regular, and one where this is predicted by the classifier from §4.1. All variants use the same

model	gold regularity	events		participants	
		macro F1	micro F1	macro F1	micro F1
Ostermann et al. (2017)	✓	58.1	66.0	n/a	n/a
Zhai et al. (2021)	X	75.1	85.7	80.3	90.3
Bare XLNet	X	40.2	53.2	39.3	60.5
w/o dep, coref, specialized	X	$46.0_{\pm 2.8}$	$58.4_{\pm 2.7}$	$47.5_{\pm 2.6}$	**75.7**$_{\pm 1.8}$
w/o dep, coref	X	$48.6_{\pm 5.2}$	$62.7_{\pm 3.7}$	$44.5_{\pm 3.3}$	$71.8_{\pm 2.1}$
w/o dep	X	$51.0_{\pm 3.7}$	$66.8_{\pm 4.3}$	**52.0**$_{\pm 3.7}$	$74.8_{\pm 2.9}$
Full model	X	**53.4**$_{\pm 1.8}$	**68.1**$_{\pm 2.3}$	$51.7_{\pm 1.6}$	$74.4_{\pm 1.4}$
Bare XLNet	✓	43.1	51.6	43.9	61.0
w/o dep, coref, specialized	✓	$46.1_{\pm 1.9}$	$55.4_{\pm 2.2}$	$51.1_{\pm 2.7}$	**75.3**$_{\pm 1.3}$
w/o dep, coref	✓	$55.3_{\pm 2.8}$	$65.8_{\pm 2.8}$	$52.5_{\pm 3.1}$	$73.6_{\pm 2.1}$
w/o dep	✓	$56.7_{\pm 3.3}$	$67.4_{\pm 3.7}$	**53.6**$_{\pm 2.9}$	$74.2_{\pm 2.0}$
Full model	✓	**57.6**$_{\pm 1.3}$	**68.1**$_{\pm 1.3}$	$52.8_{\pm 1.4}$	$73.7_{\pm 1.4}$

Table 1: Results averaged from ten-fold cross validation over five training runs. These quantities are the Hungarian versions of F1 defined in §5.1. Some models train and inference according to the regularity annotations in InScript, instead of the predictions of our regular candidate identifier. Ostermann et al. and Zhai et al. use a data split where the models see the test scenario during training; the other variants use the zero-shot data split described in §3.

number of trials for hyperparameter tuning. Afterwards, we do 5 parallel training sessions to test the models' robustness against random initializations.

6 Results

The results are shown in Table 1. All variants of our model outperform clustering based on raw XLNet embeddings by a considerable margin. Our model also performs on par with Ostermann's, although we do not have access to scenario-specific supervision whereas Ostermann's does, and our model additionally performs participant parsing. In general, we obtain a higher micro-F1 for participants than for events. This is due to the more skewed distribution of the sizes of the participant class sizes than those of the event classes.

The model extensions boost parsing accuracy significantly. Access to coreference information improves participant parsing performance. Dependency information grants a performance boost in event parsing. A closer inspection shows that with dependency information, the parser is better at grouping together event candidates with different verbs but share arguments. For example, event *sink into water* in TAKING A BATH could be evoked by *slide into water*, *sink into water*, *slip into the tub*, *lower into the tub*, etc. The verbs in these event candidates all share arguments *I* and *water* or *tub*, which our parser correctly clusters together. Without dependency information, the parser mostly group together candidates whose predicate is 'sink', the most frequent verbalization of the event.

The accuracy of our script parser differs from fold to fold. For example, we get 70.1 micro-F1 for participant parsing on TAKING A BATH, but only 43.8 on BORROWING A BOOK FROM LIBRARY. These differences result from two factors. (1) Generalization from the training scenario to the test scenario. Script parsing sometimes benefit from scenario-specific knowledge (more on this in §7.3.1), thus generalization is easier when the test scenario is more similar to the training scenarios. For example, TAKING A TRAIN would be more informative to TAKING A FLIGHT than to FIXING A FLAT TIRE. (2) Differences in the qualities of the original annotation among different scenarios.

7 Further analysis

7.1 Temporal script graphs

The events in each scenario are partially ordered with respect to their temporal order: one can only *grab a shopping cart* after *arriving at grocery store*, whereas *go to the meat section* and *go to the cheese section* can be done in arbitrary order. As mentioned above, Regneri et al. (2010) use temporal script graphs to represent the typical temporal ordering of the events in a scenario.

Given the clustering results, we can consider the stories in the test scenario as "annotated" and induce temporal script graphs for unseen scenarios. We establish temporal order as follows: event e_1 **precedes** event e_2 iff in stories where they both occur, the proportion where e_1 takes place before e_2 is beyond a threshold ζ. We expand the precedence relation to its transitive closure afterwards. Finally, if neither e_1 precedes e_2 nor e_2 precedes e_1, we

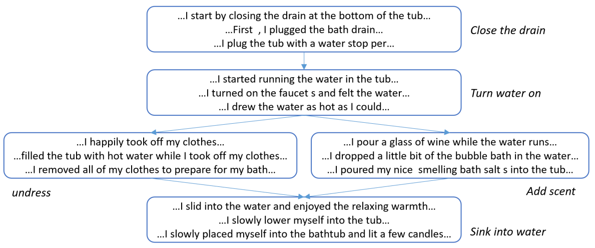

...I start by closing the drain at the bottom of the tub...
...First , I plugged the bath drain...
...I plug the tub with a water stop per...
Close the drain
...I started running the water in the tub...
...I turned on the faucet s and felt the water...
...I drew the water as hot as I could...
Turn water on
...I happily took off my clothes...
...filled the tub with hot water while I took off my clothes...
...I removed all of my clothes to prepare for my bath...
undress
...I pour a glass of wine while the water runs...
...I dropped a little bit of the bubble bath in the water...
...I poured my nice smelling bath salt s into the tub...
Add scent
...I slid into the water and enjoyed the relaxing warmth...
...I slowly lower myself into the tub...
...I slowly placed myself into the bathtub and lit a few candles...
Sink into water

Figure 4: A part of the temporal script graph for TAKING A BATH inferred from our parsing result. Each node illustrates 3 random candidates from the cluster. The event classes that the Hungarian algorithm assigned to each of these clusters are shown on the side. Further edges that could be inferred by transitivity are omitted. We see one could either *undress* first or *add scent* (to the bath tub) first before *sink into water*.

decide they could follow arbitrary order.

Our clustering results yields 75 points F1 score on the identification of temporally ordered pairs, when evaluated against the results learned from annotations in InScript. See Fig. 4 for an example. Observe that the model has learned that each event can be expressed in many different ways that are semantically similar only in the context of the scenario.

7.2 Probing

We conjectured above that the transformation φ was needed to distill the relevant information for script parsing out of the pretrained XLNet embeddings. We investigate whether this is true by freezing the embeddings $\varphi(f(c))$ (zero-shot) and the pretrained embeddings $f(c)$ (XLNet) and training models for a variety of NLP tasks that take these embeddings as input.

We probe with the following tasks. (1) Part of speech tagging and (2) named entity recognition; these mostly depend on the token itself and its local context. (3) Noun phrase chunking, which is determined by sentence-level syntax. (4) Sentence ordering, where we randomly shuffle the order of the sentences in a story and train a binary classifier to detect whether the story is shuffled. The task would need information across the entire story to conduct. POS tagging, NER and chunking are formulated as sequence labelling tasks; sentence ordering is a binary classification task. For all these tasks, a respective linear classifier maps the embeddings to the predictions. The experiments are conducted on InScript, with the same data split as is used to train our representation. InScript includes POS annotations; for NER and chunking, the labels are generated with Spacy (Honnibal and Montani, 2017, model *en_core_web_trf*). In each of these probing tasks, both representations use the same amount of GPU budget. See Fig. 5 for the results.

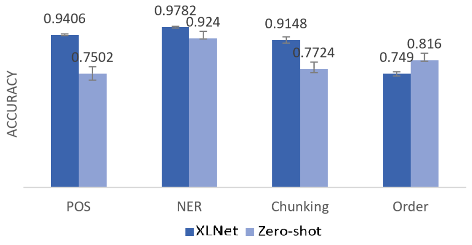


Figure 5: Performance on probing tasks. Our representation clearly favours the sentence ordering task. The error bars show one standard deviation. All differences between these pairs are significant at $\alpha = 0.05$ according to independent T-test.

The transformed representations $\varphi(f(c))$ incur performance drops on most tasks, compared to the general-purpose embeddings $f(c)$. However, the performance on sentence ordering sees a significant improvement. This supports our hypothesis that φ amplifies higher level features, which are more important to script parsing than to generic language modelling. In comparison, lower-level information about morphology and syntax is deemphasized.

7.3 Error analysis

As a side product of evaluating Hungarian F1s, we get the optimal assignment of output clusters to gold candidate classes, which equivalently labels each candidate with a event/participant class. For example, in figure 4, the candidates in the boxes are assigned to the gold class labeled aside. Errors are cases where this assignment is different that its original annotation. We manually inspected 30% of the test errors made by our full model with predicted regularity case by case and categorized them (see Table 2 for a breakdown).

7.3.1 Events

Granularity Many events could be divided into multiple sub-events, forming a hierarchy of events. This fact manifested itself into various types of errors. To begin with, the set of event labels in InScript often consists of events of different granularities. For example, in the TAKING A BATH scenario, we have *prepare for bath*, *undress* and *grab a towel*. For many event candidates, this situation renders multiple cluster assignments feasible (e.g. ... *I **took** a clean towel with me* ... in either *prepare for bath* or *grab a towel*), which not only confuses our parser, but also, judging from the corpus per se, confused many annotators of In-

Type	Granularity	Shared Verb/Arguments	#Clusters	Scenario-specific	Large Clusters	The Rest
Event	66.7%	7.8%	9.8%	15.6%	n/a	0
Participant	16.3%	n/a	11.0%	16.5%	61.5%	9.5%

Table 2: A breakdown of the parsing error types made by *full*. For a small proportion of errors we were not able to spot an obvious cause.

Script. As a result, the parser sometimes confuses one event cluster with another that includes it, or group together different events that actually fit together (*turn on water* and *fill tub with water*). In recognition of a same cause, we classify all these errors into *Granularity*.

Shared predicate or argument Some wrongly clustered events share the verb or some arguments with another class, especially when light verbs are involved, which makes the distinction harder. For example, in TAKING A TRAIN, a few *get ticket* events (e.g. "I took the ticket from him") are predicted as *conductor checks ticket* (e.g. "I gave the ticket to him").[2]

Number of clusters As the test scenario is unseen, our parser does not know how many clusters there should be, but rather terminates the clustering process with a similarity threshold. Therefore, we often end up having a different number of clusters than the corpus, which results in larger classes being split into more than one or multiple smaller classes being merged into one. For example, in the TAKING A TRAIN scenario, the model yields 14 event clusters whereas there are 15 in the corpus. In such cases, the parser has to compromise, causing damage to its performance. As a result, no predicted cluster was assigned to the *door opens* event, which refers to the door of the train opening before passengers board it. Instead, 5 out of its 6 instances end up in the same cluster that mostly comprises instances of the *get on the train* event, a fair compromise.

Scenario-specific knowledge Some candidates are only equivalent when conditioned on the scenario. For example, the *spend time in the train* event collects activities like *purchasing a coffee*, *taking a nap*, *looking out of the window*, *settling with a book*, etc. These instances are hard; our parser can address these instances as it has access to the entire event chain of the story (so it knows all these events happen between *take a seat* and *get off the train*). But the signal is weaker, which incurs weaker performance.

[2]The dependency term moderately magnifies this issue, as with it the parser tend to cluster candidates that share similar dependents together. But this is outweighed by the performance gain it grants.

7.3.2 Participants

Participant parsing also sees errors resulting from granularity, number of clusters and scenario-specific knowledge, all of which mirrors the situation of event parsing. Yet errors in participant parsing present a different landscape: most errors come from the existence of large participant classes that end up forming more that one clusters.

For example, there are two output clusters that collects instances of *train station*: one contains references to the departing station whereas the other contains references to the destination train station. Instances of participant *train* also yield two clusters: one collects references to trains in general whereas the other collects references to the train after the passenger boards it. These errors have to do with agglomerative clustering not being good at clusters whose sizes adopt a skewed distribution.

8 Conclusion

We have presented the first approach to script parsing without scenario-specific knowledge. We do this by clustering specialized word representations which are trained by optimizing cluster consistency; the model is further improved by the use of coreference and event-participant dependency information. The model greatly outperforms a baseline with general-purpose word embeddings, and performs on par with an earlier supervised model.

Our model makes it possible, for the first time, to label large quantities of unannotated data with script information. On InScript, it predicts event structures which are quite accurate with respect to temporal ordering. However, InScript is a quite specific type of text, in which all the sentences are simple and pertinent to the scenario, and they are already in the correct temporal order. In the future, we will therefore extend our approach to parsing naturally occurring text.

References

Simon Ahrendt and Vera Demberg. 2016. Improving event prediction by representing script participants. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 546–551.

Pratyay Banerjee and Chitta Baral. 2020. Self-supervised knowledge triplet learning for zero-shot question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 151–162, Online. Association for Computational Linguistics.

Avron Barr and Edward Feigenbaum. 1981. *The Handbook of Artificial Intelligence: Volume 2*. William Kaufman Inc, Los Altos, CA.

James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305.

Nathanael Chambers and Dan Jurafsky. 2008. Unsupervised learning of narrative event chains. *Proceedings of ACL-08: HLT*, pages 789–797.

Nathanael Chambers and Dan Jurafsky. 2009. Unsupervised learning of narrative schemas and their participants. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2*, pages 602–610. Association for Computational Linguistics.

Xiangyu Duan, Mingming Yin, Min Zhang, Boxing Chen, and Weihua Luo. 2019. Zero-shot cross-lingual abstractive sentence summarization through teaching generation and attention. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3162–3172, Florence, Italy. Association for Computational Linguistics.

Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew Peters, Michael Schmitz, and Luke S. Zettlemoyer. 2017. Allennlp: A deep semantic natural language processing platform.

Matthew Honnibal and Ines Montani. 2017. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To appear.

Bram Jans, Steven Bethard, Ivan Vulić, and Marie Francine Moens. 2012. Skip n-grams and ranking functions for predicting script events. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 336–344, Avignon, France. Association for Computational Linguistics.

Melvin Johnson, Mike Schuster, Quoc V. Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda Viégas, Martin Wattenberg, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2017. Google's multilingual neural machine translation system: Enabling zero-shot translation. *Transactions of the Association for Computational Linguistics*, 5:339–351.

Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *ICLR (Poster)*.

Harold W Kuhn. 1955. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97.

I-Ta Lee, Maria Leonor Pacheco, and Dan Goldwasser. 2020. Weakly-supervised modeling of contextualized event embedding for discourse relations. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4962–4972, Online. Association for Computational Linguistics.

Ashutosh Modi, Tatjana Anikina, Simon Ostermann, and Manfred Pinkal. 2016. Inscript: Narrative texts annotated with script information. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 3485–3493.

Ashutosh Modi and Ivan Titov. 2014. Inducing neural models of script knowledge. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning*, pages 49–57, Ann Arbor, Michigan. Association for Computational Linguistics.

Ashutosh Modi, Ivan Titov, Vera Demberg, Asad Sayeed, and Manfred Pinkal. 2017. Modeling semantic expectation: Using script knowledge for referent prediction. *Transactions of the Association for Computational Linguistics*, 5:31–44.

Simon Ostermann, Michael Roth, Stefan Thater, and Manfred Pinkal. 2017. Aligning script events with narrative texts. In *Proceedings of the 6th Joint Conference on Lexical and Computational Semantics (* SEM 2017)*, pages 128–134.

Ngoc-Quan Pham, Jan Niehues, Thanh-Le Ha, and Alexander Waibel. 2019. Improving zero-shot translation with language-independent constraints. In *Proceedings of the Fourth Conference on Machine Translation (Volume 1: Research Papers)*, pages 13–23, Florence, Italy. Association for Computational Linguistics.

Jerin Philip, Alexandre Berard, Matthias Gallé, and Laurent Besacier. 2020. Monolingual adapters for zero-shot neural machine translation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4465–4470, Online. Association for Computational Linguistics.

Karl Pichotta and Raymond Mooney. 2014. Statistical script learning with multi-argument events. In *Proceedings of the 14th Conference of the European*

Chapter of the Association for Computational Linguistics, pages 220–229, Gothenburg, Sweden. Association for Computational Linguistics.

Michaela Regneri, Alexander Koller, and Manfred Pinkal. 2010. Learning script knowledge with web experiments. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 979–988, Uppsala, Sweden. Association for Computational Linguistics.

Michaela Regneri, Alexander Koller, Josef Ruppenhofer, and Manfred Pinkal. 2011. Learning script participants from unlabeled data. In *Proceedings of the International Conference Recent Advances in Natural Language Processing 2011*, pages 463–470.

Rachel Rudinger, Pushpendre Rastogi, Francis Ferraro, and Benjamin Van Durme. 2015. Script induction as language modeling. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1681–1686, Lisbon, Portugal. Association for Computational Linguistics.

Roger C Schank and Robert P Abelson. 1977. *Scripts, plans, goals, and understanding: An inquiry into human knowledge structures*. Psychology Press.

Lilian DA Wanzare, Alessandra Zarcone, Stefan Thater, and Manfred Pinkal. 2016. Descript: A crowdsourced database of event sequence descriptions for the acquisition of high-quality script knowledge. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 3494–3501.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in neural information processing systems*, pages 5753–5763.

Wenpeng Yin, Jamaal Hay, and Dan Roth. 2019. Benchmarking zero-shot text classification: Datasets, evaluation and entailment approach. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3914–3923, Hong Kong, China. Association for Computational Linguistics.

Fangzhou Zhai, Vera Demberg, and Alexander Koller. 2020. Story generation with rich details. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 2346–2351, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Fangzhou Zhai, Vera Demberg, Pavel Shkadzko, Wei Shi, and Asad Sayeed. 2019. A hybrid model for globally coherent story generation. In *Proceedings of the Second Workshop on Storytelling*, pages 34–45, Florence, Italy. Association for Computational Linguistics.

Fangzhou Zhai, Iza Škrjanec, and Alexander Koller. 2021. Script parsing with hierarchical sequence modelling. In *Proceedings of* SEM 2021: The Tenth Joint Conference on Lexical and Computational Semantics*, pages 195–201.

Biao Zhang, Philip Williams, Ivan Titov, and Rico Sennrich. 2020. Improving massively multilingual neural machine translation and zero-shot translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1628–1639, Online. Association for Computational Linguistics.

A Implementation and Optimization

The model was implemented with AllenNLP 1.2 Gardner et al. (2017). The pre-trained XLNet model we used was xlnet-base-cased (https://github.com/zihangdai/xlnet/). The training is also regularized with weight decay. The optimization is performed with adam (Kingma and Ba, 2015) in conjunction with early-stopping which monitors validation loss, and the hyper-parameter tuning is performed with random hyper-parameter search (Bergstra and Bengio, 2012). Optimization takes on average 5 hours on a singe Tesla v100. We performed 20 trial for choosing the hyper-parameters.

The implementations of agglomerative clustering and Hungarian algorithm are from the *scipy* library. Table 3 shows the hyper-parameters for the best performing single run on each fold (full model, predicted regularity).

B Examples

Table 4 illustrates a couple of sample clusters. The candidates vary in their surface forms.

fold	lr	weight decay	λ_{ei}	λ_{pi}	λ_p	σ_1	σ_2	λ_c	λ_d
0	1.10E-05	0.000155	1.03	0.0147	0.343	0.887	0.263	0.00372	0.00795
1	2.06E-05	0.00677	0.457	0.882	0.125	0.392	0.391	0.0101	0.0485
2	0.000126	0.000896	0.0105	0.0148	0.0144	0.903	0.728	0.00533	0.0118
3	2.91E-05	0.000166	0.0237	0.0649	0.256	0.373	0.544	0.0619	0.00867
4	7.08E-05	0.00051	0.15	0.0711	0.745	0.681	0.562	0.0121	0.0216
5	0.000353	0.00121	1.51	0.0371	0.00211	0.171	0.692	0.0872	0.016
6	0.000423	1.07E-05	0.204	0.018	0.177	0.113	0.632	0.242	0.00517
7	7.88E-05	0.00303	0.734	0.0136	0.517	0.189	0.948	0.0047	0.111
8	2.72E-05	7.32E-05	0.568	0.00151	0.0589	0.676	0.566	0.00424	0.0475
9	2.09E-05	5.19E-05	0.0327	0.555	0.243	0.788	0.95	0.0866	0.0119

Table 3: Hyper-parameters

ground truth	text
turn water on bath	... I might drain the tub and **put** in more water ...
sink into water	... I turn off the faucet and **sink** into bliss ...
sink into water	... Then I **slid** into the water and enjoyed the relaxing warmth for twenty or more minutes...
sink into water	... I gingerly **lowered** myself into the nice warm water and immediately began to relax...
sink into water	... I eased my way into the tub and let myself **sink** into the water ...
sink into water	... I slowly **sunk** the rest of my body , and closed my eyes...
sink into water	... the tub was full and ready . I **slipped** into the tub and soaked in the bliss...
washing tools	...then I lather up with either **soap** or shower gel ...
water	...After I scrub really good and finish singing , I pour **water** continuously on my body...
washing tools	...I pour water continuously on my body until all the **soap** was he s off...
washing tools	...I cleaned my hair with some **shampoo** and washed my body with a wash cloth and rinsed...
washing tools	...shampooed my hair and applied some **conditioner** then washed my body...
washing tools	...applied some condition er then washed my body using some **liquid body wash**...
washing tools	...After I have washed everything , I rinse the **soap** from my body with the water in the tub...
washing tools	...on the corner of the bathtub . I lather ed **it** up and washed my arms , my legs...
washing tools	...take a wash cloth and **soap** or body wash to give yourself a good scrub down...
washing tools	...You can put the **shampoo** in your hair...
washing tools	...place your head under the faucet to rinse out the **soap** . Enjoy your bath !
washing tools	...washed myself with a **wash cloth** and soap . Then I leaned my head against...
washing tools	...stepped into the bath tub . I used **soap** and a wash cloth to clean myself...

Table 4: Example output clusters. Top: event; bottom: participant. The table presents a random selection of instances from these clusters as the original output could contain hundreds of instances.