Guided Neural Language Generation for Automated Storytelling

Prithviraj Ammanabrolu, Ethan Tien, Wesley Cheung,
Zhaochen Luo, William Ma, Lara J. Martin, and Mark O. Riedl
School of Interactive Computing
Georgia Institute of Technology
Atlanta, GA, USA
{raj.ammanabrolu, etien, wcheung8,
zluo, wma61, ljmartin, riedl}@gatech.edu

Abstract

Neural network based approaches to automated story plot generation attempt to learn how to generate novel plots from a corpus of natural language plot summaries. Prior work has shown that a semantic abstraction of sentences called *events* improves neural plot generation and and allows one to decompose the problem into: (1) the generation of a sequence of events (event-to-event) and (2) the transformation of these events into natural language sentences (event-to-sentence). However, typical neural language generation approaches to event-to-sentence can ignore the event details and produce grammatically-correct but semantically-unrelated sentences. We present an ensemble-based model that generates natural language guided by events. Our method outperforms the baseline sequence-to-sequence model. Additionally, we provide results for a full end-to-end automated story generation system, demonstrating how our model works with existing systems designed for the event-to-event problem.

1 Introduction

Automated story plot generation is the problem of creating a sequence of main plot points for a story in a given domain. Generated plots must remain consistent across the entire story, preserve long-term dependencies, and make use of commonsense and schematic knowledge (Wiseman et al., 2017). Early work focused on symbolic planning and case-based reasoning (Meehan, 1977; Turner and Dyer, 1986; Lebowitz, 1987; Gervás et al., 2005; Porteous and Cavazza, 2009; Riedl and Young, 2010; Ware and Young, 2011; Farrell et al., 2019) at the expense of manual world domain knowledge engineering. Neural-network–based approaches to story and plot generation train a neural language model on a corpus of stories to predict the next character, word, or sentence in a sequence based on a history of tokens (Jain et al., 2017; Clark et al., 2018; Fan et al., 2018; Martin et al., 2018; Peng et al., 2018; Roemmele, 2018).

The advantage of neural network based approaches is that there is no need for explicit domain modeling beyond providing a corpus of example stories. The primary pitfall of neural language model approaches to story generation is that the space of stories that can be generated is huge, which in turn, implies that in a textual story corpora any given sentence will likely only be seen once. Martin et al. (2018) propose the use of a semantic abstraction called *events*, demonstrating that it aids in reducing the sparsity of the dataset. They define an event to be a unit of a story that creates a world state change; specifically, an event is a tuple containing a subject, verb, direct object, and some additional disambiguation tokens.

The event representation enables the decomposition of the plot generation task into two sub-problems: *event-to-event* and *event-to-sentence*. Event-to-event is broadly the problem of generating the sequence of events that together comprise a plot. A model for addressing this problem is also responsible for maintaining plot coherence and consistency. These events are abstractions and aren't human-readable. Thus the second sub-problem, event-to-sentence, focuses on transforming these events into natural language sentences. This second sub-problem can also be viewed as *guided language generation*, using a generated event as a guide.

Martin et al. (2018) further propose that this latter event-to-sentence problem can be thought of as a translation task—translating from the language of events into natural language. We find, however, that the sequence-to-sequence LSTM networks (Sutskever et al., 2014) that they chose to address the problem frequently ignore the input event and only generate text based on the orig-

Proceedings of the Second Storytelling Workshop, pages 46–55
Florence, Italy, August 1, 2019.

inal corpus, overwriting the plot-based decisions made during event-to-event. There are two contributing factors. Firstly, event-to-event models tend to produce previously-unseen events, which, when fed into the event-to-sentence model results in unpredictable behavior. The mapping from an unseen event to a sentence is unknown to a basic sequence-to-sequence model. Secondly, sentences are often only seen once in the entire corpus. Despite being converted into events, the sparsity of the data means that each event is still likely seen a very limited number of times.

The contributions of the paper are thus twofold. We present an ensemble-based system for event-to-sentence that allows for guided language generation and demonstrate that this outperforms a baseline sequence-to-sequence approach. Additionally, we present the results of a full end-to-end story generation pipeline as originally proposed by Martin et al. (2018) (Figure 1), showing how all of the sub-systems can be integrated.

2 Related Work

Early storytelling systems were based on symbolic planning (Pérez y Pérez and Sharples, 2001; Riedl and Young, 2010; Meehan, 1977; Lebowitz, 1987; Ware and Young, 2011) and case-based reasoning (Turner and Dyer, 1986; Pérez y Pérez and Sharples, 2001; Gervás et al., 2005). These systems required a high knowledge-engineering overhead in terms of operators or stories transcribed into symbolic form. Consequently, these systems were only capable of generating stories in relatively limited domains.

Machine learning approaches attempt to learn domain information from a corpus of story examples (Swanson and Gordon, 2012; Li et al., 2013). Recent work has looked at using recurrent neural networks (RNNs) for story and plot generation. Roemmele and Gordon (2018) use LSTMs with skip-though vector embeddings (Kiros et al., 2015) to generate stories. Khalifa et al. (2017) train an RNN on a highly specialized corpus, such as work from a single author.

Fan et al. (2018) introduce a form of hierarchical story generation in which a premise is first generated by the model and is then transformed into a passage. This is a form of guided generation wherein a single sentence of guidance is given. Similarly, Yao et al. (2019) decompose story generation into planning out a storyline and then generating a story from it. Our work differs in that we use an event-to-event process that provides guidance to event-to-sentence.

3 Event-to-Event Implementation

In order to create a full improvisational storytelling pipeline, we first needed to implement an event-to-event model such that generated events can be inputted into our event-to-sentence system. Martin et al. (2018) showed that the performance on both event-to-event and event-to-sentence improves when using an abstraction known as an *event* is used instead of natural language sentences.

In our work, events are defined as a 5-tuple of $\langle s, v, p, o, m \rangle$ where v is a verb, s is the subject of the verb, o is the object, p is the corresponding preposition, and m can be a modifier, prepositional object, or indirect object—any of which can be absent. All elements are stemmed and generalized with the exception of the preposition. We follow the same generalization process as Martin et al. (2018), using enumerated named entity recognition tags, VerbNet (Schuler and Kipper-Schuler, 2005) v3.3 to generalize the verbs, and WordNet (Miller, 1995) v3.1 for the nouns.

Our event-to-event system is the policy gradient deep reinforcement learner from Tambwekar et al. (2019). Briefly, the technique starts with a sequence-to-sequence LSTM model trained to perform the event-to-event task. It is trained on a sequence of "eventified" plot summaries. Using the REINFORCE algorithm (Williams, 1992), we backpropagate a reward based on how close the generated event is to a pre-trained goal. Here, we are using genre-appropriate verbs (specifically, VerbNet classes) as goals—verbs that appear often at the end of the stories in our dataset. The reward is the product of the distance of each verb from the goal verb by the normalized frequency of how often the verb occurs before the goal verb in stories. Details of how the reward is calculated are given in Tambwekar et al. (2019).

The final event-to-event network is then placed into the pipeline as the "Event-to-Event" module, seen in Figure 1, and its output is fed into the following event-to-sentence models during testing.

4 Event-to-Sentence

We define event-to-sentence to be the problem of selecting a sequence of words $s_t = s_{t_0}, s_{t_1}, ..., s_{t_k}$

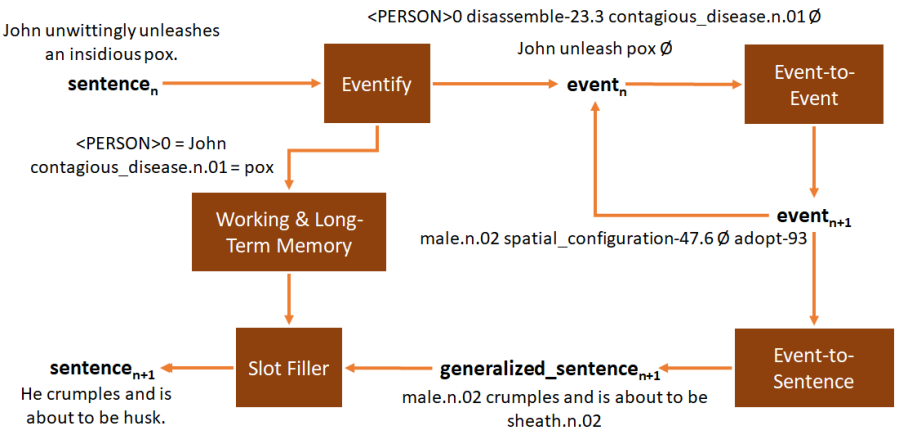

Figure 1: The full automated story generation pipeline illustrating an example where the event-to-event module generates only a single following event.

given the current input event e_t, i.e. the current sentence is generated based on maximizing $Pr(s_t|e_t;\theta)$ where θ refers to the parameters of the generative system. The eventification in Section 3 is a lossy process in which some of the information from the original sentence is dropped. Thus, the task of event-to-sentence involves filling in this missing information. There is also no guarantee that the event-to-event process will produce an event that is part of the event-to-sentence training corpus, simply due to the fact that the space of potential generated events is very large; the correct mapping from the generated event to a natural language sentence would be unknown.

In prior work, Martin et al. (2018) use a sequence-to-sequence LSTM neural network to translate events into sentences. Even with the "split and pruned" sentences that they create—which we also use (see Section 5)—we find that this vanilla sequence-to-sequence technique is not robust to the afore-mentioned challenges. We observe that a sequence-to-sequence network ends up operating as a simple language model and often ignores the input event when generating a sentence. The generated sentence is usually grammatically correct, but retains little of the semantic meaning given by the event.

We thus look for other forms of guided neural language generation, with the goals of preserving the semantic meaning from the event in addition to keeping the generated sentences interesting. We propose four different models—optimized towards these two objectives, and a baseline fifth model that is used as a fallthrough: (1) a retrieve-and-edit model based on Hashimoto et al. (2018); (2) template filling; (3) sequence-to-sequence with Monte Carlo beam decoding; (4) sequence-to-sequence with a finite state machine decoder; and (5) vanilla sequence-to-sequence. We find that none of these models by themselves can successfully find a balance between the goals of *retaining all of the event tokens* and *generating interesting output*. However, each of the models possess their own strengths and weaknesses—each model essentially being optimized towards a different point on the spectrum between the two goals. We thus combine these models into an ensemble in an attempt to minimize the weaknesses of each individual model and achieve a balance between retaining semantic meaning from the event and generating interesting sentences.

In all of the following experiments, the task is to translate events into "generalized" sentences. A generalized sentence is one in which nouns are replaced by WordNet Synsets.

4.1 Retrieve-and-Edit

The first model is based on the retrieve-and-edit framework for predicting structured outputs (Hashimoto et al., 2018), which we will refer to as *RetEdit*. We first learn a task-specific similarity between event tuples by training an encoder-decoder to map each event onto an embedding that can reconstruct the output sentence; this is our retriever model. Next, we train an editor model which maximizes the likelihood of generating the target sentence given both the input event and a retrieved event-sentence example pair. We used a standard sequence-to-sequence model with attention and copying (Gu et al., 2016) to stand in

as our editor architecture. Although this framework was initially applied to the generation of GitHub Python code and Hearthstone cards, we extend this technique to generate sentences from our event tuples. Specifically, we first initialize a new set of word embeddings with GLoVe (Pennington et al., 2014), using random initialization for out-of-vocabulary words. We use our training set to learn weights for the retriever and editor models, set confidence thresholds for the model with the validation set, and evaluate performance using the test set.

In order to generate a sentence from a given input event, there are two key phases: the "retrieve" phase and the "edit" phase. With respect to the input event, we first retrieve the nearest-neighbor event and its corresponding sentence in the training set using the retriever model. Then, passing both the retrieved event-sentence pair and the input event as inputs, we use the editor model to generate a sentence using beam search. Many of the successes produced by the model stem from its ability to retain the complex sentence structures that appear in our training corpus. However, this interaction with the training data can also prove to be a major drawback of the method; target events that are distant in the embedding space from training examples typically result in poor sentence quality.

Since RetEdit relies heavily on having good examples, we set the confidence of the retrieve-and-edit model to be proportional to $1- \ retrieval \ distance$ when generating sentences as a lower retrieve distance implies greater confidence. We also observe that our mapping from event to sentence is not a one-to-one function. There are occasionally multiple sentences that map to a single event, resulting in retrieval distances of 0.In this case, the example sentence is returned without any modifications.

4.2 Sentence Templating

As mentioned earlier, the baseline sequence-to-sequence network operates as a simple language model and can often ignore the input event when generating a sentence. However, we know that our inputs, an event tuple will have known parts of speech.We created a simplified grammar for the syntax of sentences generated from events:

$$\begin{aligned} S &\rightarrow NP \ \ v \ \ (NP) \ \ (PP) \\ NP &\rightarrow d \ \ n \\ PP &\rightarrow p \ \ NP \end{aligned}$$

where d is a determiner that will be added and the rest of the terminal symbols correspond to an argument in the event, with n being s, o, or m, depending on its position in the sentence. The resulting sentence would be $[\_\_ \ s]\{v \ [\_\_ \ o] \ [p \ \_\_ \ m]\}$ where blanks indicate where words must be added to make a complete sentence.

First, our algorithm predicts the most likely VerbNet frame based on the contents of the input event (how many and which arguments are filled). VerbNet provides a number of syntactic structures for different verb classes based on how the verb is being used. For example, if the input event contains 2 nouns and a verb without a preposition, we assume that the output sentence takes the form of [NP V NP], but if it has 2 nouns, a verb, *and* a proposition, then it should be [NP V PP].

Second, we apply a Bidirectional LSTM (BiLSTM) language model trained on the generalized sentences in our training corpus. Given a word, we can generate words before and after it, within a particular phrase as given by some of the rules above, and concatenate the generated sentence fragments together. Specifically, we use the AWD-LSTM (Merity et al., 2018) architecture as our language model since it is currently state-of-the-art.

At decode time, we continue to generate words in each phrase until we reach a stopping condition: (1) reaching a maximum length (to prevent run-on sentences); or (2) generating a token that is indicative of an element in the next phrase, for example seeing a verb being generated in a noun phrase. When picking words from the language model, we noticed that the words "the" and "and" were extremely common. To increase the variety of the sentences, we sample from the top k most-likely next words and enforce a number of grammar-related rules in order to keep the coherence of the sentence. For example, we do not allow two determiners nor two nouns to be generated next to each other.

One can expect that many of the results will look structurally similar. However, with this approach, we can guarantee that the provided tokens in the event will appear in the generated sentence. To determine the confidence of the model for each

sentence, we sum the loss after each generated token, normalize to sentence length, and subtract from 1 as higher loss translates to lower confidence.

4.3 Monte-Carlo Beam Search

Our third method is an adaptation of *Monte Carlo Beam Search* (Cazenave, 2012) for event-to-sentence. We train a sequence-to-sequence model on pairs of events and generalized sentences. At decoding time, we run Monte Carlo beam search as an alternative search strategy within the decoder network. This method differs from traditional beam search in that it introduces another scoring term that is used to re-weigh all the beams at each timestep.

After top-scoring words are outputted by the model at each time step, playouts are done from each word, or *node*. A node is the final token of the partially-generated sequences on the beam currently and the start of a new playout. During each playout, one word is sampled from a softmax produced at each step over all words in the vocabulary. The decoder network is unrolled until it reaches the "end-of-story" tag. Then, the previously-generated sequence and the sequence generated from the current playout are concatenated together and passed into a scoring function that computes the current playout's score.

The scoring function is a combination of (1) BLEU scores up to 4-grams between the input event and generated sentence, as well as (2) a weighted 1-gram BLEU score between each item in the input event and generated sentence. The weights combining the 1-gram BLEU scores are learned during validation time where the weight for each word in the event that *does not* appear in the final generated sequence gets bumped up. Multiple playouts are done from each word and the score s for the current word is computed as:

$$s_t = \alpha * s_{t-1} + (1 - \alpha) * AVG(playout_t) \quad (1)$$

where α is a constant.

In the end, k of the partial sequences with highest playout scores are kept as the current beam. For the ensemble, this model's confidence score is the final score of the highest-scoring end node.

Monte Carlo beam search excels at creating diverse output. Since the score for each word is based on playouts that sample based on weights at each timestep, it is possible for the output to be different across runs. The Monte Carlo beam decoder has been shown to generate better sentences that are more grammatically-correct than the other techniques in our ensemble, while sticking more to the input than a traditional beam decoder. However, there is no guarantee that all input event tokens will be included in the final output sentence.

4.4 Finite State Machine Constrained Beams

Beam search in its various forms, including Monte Carlo playouts, cannot ensure that the tokens from an input event appear in the outputted sentence. As such, we adapted the algorithm to fit such lexical constraints. Anderson et al. (2016) adapted beam search to fit captions for images, with the lexical constraints coming from sets of image tags. The method they devised, which they named *Constrained Beam Search*, used finite state machines to guide the beam search toward generating the desired tokens. This approach, which we have coopted for event-to-sentence, attempts to achieve a balance between the flexibility and sentence quality typical of a beam search approach, while also adhering to the context and story encoded in the input events that more direct approaches (e.g. Section 4.2) would achieve.

The algorithm works on a per-event basis, beginning by generating a finite state machine. This finite state machine consists of states that enforce the presence of input tokens in the generated sentence. As an example, assume we have an n-token input event, $\{t_1, t_2, t_3, ..., t_n\}$. The corresponding machine consists of 2^n states. Each state maintains a search beam of size B^s with at most b output sequences, corresponding to the configured beam size s. At each time step, every state (barring the initial state) receives from predecessor states those output sequences whose last generated token matches an input event token. The state then adds to its beam the b most likely output sequences from those received.

In the example, generating token t_1 moves the current state from the initial state to the state corresponding to t_1, t_3 to a state for t_3, and so on. The states t_1 and t_3 then, after generating tokens t_1 and t_3 respectively, transmit said sequences to the state $t_{1,3}$. The states and transitions proceed as such until reaching the final state, wherein they have matched every token in the input event. Completed sequences in the final state contain all input event tokens, thus providing us with the ability to

retain the semantic meaning of the event.

As much as the algorithm is based around balancing generating good sentences with satisfying lexical constraints, it does not perform particularly well at either. It is entirely possible, if not at all frequent, for generated sentences to contain all input tokens but lose proper grammar and syntax, or even fail to reach the final state within a fixed time horizon. This is exacerbated by larger tuples of tokens, seen even at just five tokens per tuple. To compensate, we relax our constraint to permit output sequences that have matched at least three out of five tokens from the input event. Still, at least some of the generated sentences will exhibit the problems mentioned above.

4.5 Ensemble

The entire event-to-sentence ensemble is designed as a cascading sequence of the four models above. We use the confidence scores generated by each of the models in order to re-rank the outputs of the individual models. This is done by setting a confidence threshold for each of the models such that if a confidence threshold fails, the next model in the cascade is tried. The thresholds are tuned by measuring the confidence scores generated on the validation set of the corpus. The cascading sequence is defined in the order that the individual models are presented above: (1) retrieve-and-edit, (2) sentence templating, (3) Monte Carlo beam search, (4) finite state constrained beam search, and (5) standard beam search. This structure also saves on computation as it sequentially queries each model, terminating early and returning an output sentence if the confidence threshold for any of the individual models are met.

The event first goes through the retrieve-and-edit framework, which generates a sentence and corresponding confidence score. We observe that this framework performs well when it is able to retrieve a sample from the training set that is relatively close in terms of retrieval distance to the input. Given the sparsity of the dataset, this happens with a relatively low probability, and so we place this model first in the sequence.

The next two models are each optimized towards one of our two main goals. The sentence templating approach retains all of the tokens within the event and so loses none of its semantic meaning, at the expense of generating a more interesting sentence. The Monte-Carlo approach, on the other hand, makes no guarantees regarding retaining the original tokens within the event but is capable of generating a diverse set of sentences. We thus cascade first to the sentence templating model and then the Monte-Carlo approach, implicitly placing greater importance on the goal of retaining the semantic meaning of the event.

The final model queried is the finite state machine constrained beam search. This model has no confidence score; either the model is successful in producing a sentence within the given length with the event tokens or not.In the case that the finite state machine based model is unsuccessful in producing a sentence, the final fallthrough model—the baseline sequence-to-sequence model with standard beam search decoding—is used.

5 Dataset

To aid in the performance of our story generation, we select a single genre: science fiction (sci-fi). We scraped long-running science fiction TV show plot summaries from the fandom wiki service *wikia.com*. This fandom wiki service contains longer and more detailed plot summaries than the dataset used in Tambwekar et al. (2019), both of which are qualities that we believe to be important for the overall story generation process. The corpus contains 2,276 stories in total, each an episode of a TV show. The average story length is 89.23 sentences. There are stories from 11 shows, with an average of 207 stories per show, from shows like *Doctor Who*, *Futurama*, and *The X-Files*. The data was pre-processed to simplify alien names in order to aid named entity recognition.

Then the sentences were split, partially following the "split-and-pruned" methodology of Martin et al. (2018). Sentences were split at S-bars and conjunctions separating S's, and the subject of the sentence was re-inserted in the new sentences. Once the sentences were split, they were "eventified" as described in Section 3. One benefit of having split sentences is that there is a higher chance of having a 1:1 correspondence between sentence and event, instead of a single sentence becoming multiple events. After the data is fully prepared, it is split in a 8:1:1 ratio to create the training, validation, and testing sets respectively.

6 Experiments

We perform two sets of experiments, one set evaluating our models on the event-to-sentence prob-

lem by itself, and another set intended to evaluate the full storytelling pipeline.

Each of the models in the event-to-sentence ensemble are trained on the training set in the sci-fi corpus. The exact training details for each of the models are as described above. Note that we present results for the generalized sentences instead of the sentences after slot-filling, as shown in Figure 1, to directly measure the output of the event-to-sentence ensemble. Additionally, all of the models in the ensemble slot-fill the verb automatically—filling a VerbNet class with a verb of appropriate conjugation—except for the sentence templating model which does verb slot-filling during post-processing.

After the models are trained, we pick the cascading thresholds for the ensemble by running the validation set through each of the models and generating confidence scores. This is done by running a grid search through a limited set of thresholds such that the overall BLEU-4 score (Papineni et al., 2002) of the generated sentences in the validation set is maximized. These thresholds are then frozen when running the final set of evaluations on the test set. For the baseline sequence-to-sequence method, we decode our output with a beam size of 5. We report perplexity and BLEU-4 scores, comparing against the gold standard from the test set. Perplexity is a measure of the predictive accuracy of a model and is calculated as:

$$Perplexity = 2^{-\sum_x p(x) \log_2 p(x)} \quad (2)$$

where x is a token in the text, and

$$p(x) = \frac{count(x)}{\sum_{y \in Y} count(y)} \quad (3)$$

where Y is the vocabulary. Our BLEU-4 scores are naturally low (where higher is better) because of the creative nature of the task—good sentences may not use any of the ground-truth n-grams.

The second experiment uses event sequences generated by our event-to-event system as summarized in Section 3. Our event-to-event system requires goals in the form of verb classes. For the science fiction data, common endings for stories are VerbNet classes like "escape-51.1", "discover-84", and "get-13.5.1". In this paper, we will use the goal "discover-84". We seed the event-to-event system with events extracted from the first sentences of stories found in the test set. The system then generates a sequence of events until it reaches the goal verb. We then present this sequence of events as well as the corresponding generalized sentences generated using our ensemble.

Model	Perplexity	BLEU	Length
RetEdit	71.354	0.041	9.27
Templates	203.629	0.0034	5.43
Monte Carlo	71.385	0.0453	7.91
FSM	104.775	0.0125	10.98
Seq2seq	103.176	0.0425	6.59
Ensemble	**70.179**	**0.0481**	9.22

Table 1: Perplexity and BLEU scores along with average sentence lengths, thresholds, and utilization percentages for event-to-sentence models on the test set.

7 Results/Discussion

Model	Thresholds	Test Utilization	Pipeline Utilization
RetEdit	0.8	94.91%	55.71%
Templates	0.8	0.22%	0.91%
Monte Carlo	0.1	4.29%	41.10%
FSM	-	0.15%	0.68%
Seq2seq	-	0.43%	1.60%

Table 2: Thresholds and utilization percentages for the models on the test sets and on the events generated by the event-to-event system.

Our results are presented in the form of three tables. Table 1 shows the perplexity, BLEU-4 scores, and average sentence length for event-to-sentence on the testing set for each of the models, ensemble, and baseline. Note that some of the models, such as the sentence templates, ignore the idea of a gold standard sentence and are thus poorly optimized towards perplexity and BLEU scores. The ensemble, as predicted, still performs better than any of the individual models as it is designed to combine the models such that each of their weaknesses are minimized. The average sentence lengths highlight the differences between the models, with the templates producing the shortest sentences and the finite state machine taking longer to generate sentences due to the constraints it needs to satisfy. Table 2 shows the confidence thresholds after tuning the ensemble. The RetEdit and sentence template models need 80% confidence in their results, or the next model in the cascade is tried. Table 2 also shows how often each model in the ensemble is used generating sentences from the eventified testing corpus and from event-to-event. RedEdit was heavily used on the test set, likely due the train and test sets having a similar distribution of data. On the pipeline examples RetEdit is used much less—events generated by event-to-event may be very different from those in the training set. A majority of the events

Input Event	RetEdit	Templates	Monte Carlo	FSM	Gold Standard
⟨<PRP>, act-114-1-1, to, ∅, event.n.01⟩	<PRP> and <PERSON>0 move to the event.n.01 of the natural_object.n.01.	<PRP> act-114-1-1 to event.n.01.	<PRP> moves to the nearest natural_object.n.01.	physical_entity.n.01 move back to the phenomenon.n.01 of the craft.n.02...	<PRP> move to the event.n.01.
⟨<PERSON>2, send-11.1, through, <PERSON>6, <LOCATION>1⟩	<PERSON>2 sends <PERSON>6 through the <LOCATION>1.	The <PERSON>2 send-11.1 the <PERSON>6 through <LOCATION>1.	<PERSON>2 passes this undercover in the body_part.n.01 and collapses.	∅	In activity.n.01 to avoid <PRP> out.n.01 <PERSON>2 would transport <PERSON>6 through the <LOCATION>1.
⟨<PERSON>0, admire-31.2, ∅, <PERSON>3, ∅⟩	<PERSON>0 believes <PERSON>3.	<PERSON>0 admire-31.2 and <PERSON>3	<PERSON>0 hates <PERSON>3 saying <PRP> s not ready for duration.n.03 .	<PERSON>0 and <PERSON>0 comes in <PRP> content.n.05 for wrongdoing.n.02 and says <PERSON>0 has made on line.n.23 have made trait.n.01.	A pivotal artifact.n.01 in <PRP> act.n.02 is a examination.n.01 divised by <LOCATION>0 to make <PERSON>0 hate <PRP> <PERSON>3.

Table 3: Event-to-sentence examples for each model. ∅ represents an empty parameter; <PRP> is a pronoun.

Input Sentence	Extracted Event	Generated Events (Event-to-Event)	Generated Sentences (Event-to-Sentence; ensemble)
On Tatooine, Jabba the Hutt inspects the drone barge recently delivered to him.	⟨<ORGANIZATION>0, assessment-34.1, ∅, vessel.n.02, ∅⟩	⟨<PERSON>1, settle-36.1.2, ∅, indicator.n.03, indicator.n.03 ⟩; ⟨music.n.01, escape-51.1-1, from, ∅, ∅⟩; ⟨<PRP>, discover-84, to, run-51.3.2, progenitor.n.01⟩	The <ORGANIZATION>0 can not scan the vessel.n.02 of the <VESSEL>0. <PERSON>1 decides to be a little person.n.01 at the structure.n.01. the 'music.n.01 arrives. <PRP> finds a lonely person.n.01 on the upper one of the craft.n.02 which is not an personal_letter.n.01 but does not respond to hails .
Boba Fett has just chased down another bounty, a Rodian art dealer who sold fake works to Gebbu the Hutt.	⟨<PERSON>0, chase-51.6, ∅, bounty.n.04, ∅⟩	⟨<PERSON>0, chase-51.6, to, magnitude.n.01, ∅⟩; ⟨magnitude.n.01, comprehend-87.2, off, craft.n.02, magnitude.n.01⟩; ⟨<PERSON>2, amuse-31.1, off, ∅, ∅⟩; ⟨<PERSON>2, discover-84, off, change_of_integrity.n.01, ∅⟩	<PERSON>0 enters the bounty.n.04 and tells <PRP>. <PERSON>0 attaches the explosive.a.01 to the person.n.01 who is trying to fix the device.n.01 . the magnitude.n.01 doesn't know the craft.n.02 off the craft.n.02. <PERSON>2 is surprised when <PRP> learns that the person.n.01 is actually <PERSON>7 . <PERSON>2 sees the change_of_integrity.n.01 and tells <PRP>.

Table 4: End-to-end pipeline examples on previously-unseen input data.

that fall through RetEdit are caught by our Monte Carlo beam search, irrespective of the fact that RetEdit and sentence templates are most likely to honor the event tokens.

Table 3 presents concrete examples of the generalized sentence outputs of each of the event-to-sentence models and some trends are evident. Retrieve-and-Edit focuses on semantics at the expense of sentence quality. Sentence templates produces output that matches the input event but is very formulaic. Monte Carlo generates entertaining and grammatically-correct sentences, but occasionally loses the semantics of the input event. The finite state machine attempts to achieve a balance between semantics and generating entertaining output, however it sometimes fails to produce an output given the constraints of the state machine itself. We further present the results of an entire working pipeline in Table 4, demonstrating the event-to-sentence's ability to work with an existing plot generator.

8 Conclusions

Although event representations were shown in the past to improve performance on plot generation tasks—allowing for planning toward plot points, we are faced with the problem of translating these events into syntactically- and semantically-sound sentences that are both interesting and keep the meaning of the original event. We have found that no one model is able to fully solve this task and so we present the combination of four approaches—each of which excel at and struggle with different aspects of the translation—into an ensemble.

RetEdit excels when the input event is drawn from a similar distribution as our training set, but the FSM approach does not depend on the distribution that the input is drawn from. The Sentence Templates generate semantically-sound sentences that are generic in structure, whereas Monte Carlo beam search generates more diverse output but is not guaranteed to retain the meaning of the input event. These models lie at different points on the spectrum between retaining meaning and generating interesting sentences. Future state-of-the-art sentence-filling techniques can also be added to the ensemble to account for the weaknesses of current models. This work completes the end-to-end story generation pipeline previously-conceived by Martin et al. (2018) and serves as a stepping stone for research in sentence expansion or event-to-sentence tasks.

References

Peter Anderson, Basura Fernando, Mark Johnson, and Stephen Gould. 2016. Guided open vocabulary image captioning with constrained beam search. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, page 36945.

Tristan Cazenave. 2012. Monte Carlo Beam Search. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):68–72.

Elizabeth Clark, Yangfeng Ji, and Noah A. Smith. 2018. Neural Text Generation in Stories Using Entity Representations as Context. In *NAACL-HLT*.

Angela Fan, Mike Lewis, and Yann Dauphin. 2018. Hierarchical Neural Story Generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pages 889–898.

R. Farrell, S. G. Ware, and L. J. Baker. 2019. Manipulating narrative salience in interactive stories using indexter's pairwise event salience hypothesis. *IEEE Transactions on Games*, pages 1–1.

Pablo Gervás, Belén Díaz-Agudo, Federico Peinado, and Raquel Hervás. 2005. Story plot generation based on CBR. *Knowledge-Based Systems*, 18(4-5):235–242.

Jiatao Gu, Zhengdong Lu, Hang Li, and Victor O. K. Li. 2016. Incorporating copying mechanism in sequence-to-sequence learning. *Association for Computational Linguistics (ACL)*.

Tatsunori B Hashimoto, Kelvin Guu, Yonatan Oren, and Percy Liang. 2018. A Retrieve-and-Edit Framework for Predicting Structured Outputs. In *32nd Conference on Neural Information Processing Systems (NeurIPS 2018)*, Montréal, Canada.

Parag Jain, Priyanka Agrawal, Abhijit Mishra, Mohak Sukhwani, Anirban Laha, and Karthik Sankaranarayanan. 2017. Story generation from sequence of independent short descriptions. In *SIGKDD Workshop on Machine Learning for Creativity (ML4Creativity)*.

Ahmed Khalifa, Gabriella A. B. Barros, and Julian Togelius. 2017. DeepTingle. In *International Conference on Computational Creativity*, page 8.

Ryan Kiros, Yukun Zhu, Ruslan R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Skip-thought vectors. In *Advances in neural information processing systems*, pages 3294–3302.

Michael Lebowitz. 1987. Planning Stories. In *Proceedings of the 9th Annual Conference of the Cognitive Science Society*, pages 234–242.

Boyang Li, Stephen Lee-Urban, George Johnston, and Mark O. Riedl. 2013. Story generation with crowdsourced plot graphs. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*, AAAI'13, pages 598–604. AAAI Press.

Lara J. Martin, Prithviraj Ammanabrolu, Xinyu Wang, William Hancock, Shruti Singh, Brent Harrison, and Mark O. Riedl. 2018. Event Representations for Automated Story Generation with Deep Neural Nets. In *Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18)*, pages 868–875, New Orleans, Louisiana.

James R. Meehan. 1977. TALE-SPIN, an interactive program that writes stories. *Proceedings of the 5th international joint conference on Artificial intelligence*, 1:91–98.

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. 2018. Regularizing and Optimizing LSTM Language Models. In *6th International Conference on Learning Representations, ICLR 2018*.

George A. Miller. 1995. WordNet: A Lexical Database for English. *Communications of the ACM*, 38(11):39–41.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics.

Nanyun Peng, Marjan Ghazvininejad, Jonathan May, and Kevin Knight. 2018. Towards Controllable Story Generation. In *Proceedings of the First Workshop on Storytelling*, pages 43–49, New Orleans, Louisiana. Association for Computational Linguistics.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Mike Pérez y Pérez and Rafael Sharples. 2001. MEXICA: A computer model of a cognitive account of creative writing. *Journal of Experimental & Theoretical Artificial Intelligence*, 13(2001):119–139.

Julie Porteous and Marc Cavazza. 2009. Controlling narrative generation with planning trajectories: The role of constraints. In *Joint International Conference on Interactive Digital Storytelling*, volume 5915 LNCS, pages 234–245. Springer.

Mark O Riedl and R Michael Young. 2010. Narrative Planning: Balancing Plot and Character. *Journal of Artificial Intelligence Research*, 39:217–267.

Melissa Roemmele. 2018. *Neural Networks for Narrative Continuation*. Ph.D. thesis, University of Southern California.

Melissa Roemmele and Andrew S Gordon. 2018. An Encoder-decoder Approach to Predicting Causal Relations in Stories. In *Proceedings of the First Workshop on Storytelling*, pages 50–59, New Orleans, Louisiana. Association for Computational Linguistics.

Karin Kipper Schuler and Karen Kipper-Schuler. 2005. *VerbNet: A Broad-Coverage, Comprehensive Verb Lexicon*. Ph.D. thesis, University of Pennsylvania.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to Sequence Learning with Neural Networks. In *Advances in Neural Information Processing Systems*, pages 3104–3112.

Reid Swanson and Andrew Gordon. 2012. Say Anything: Using textual case-based reasoning to enable open-domain interactive storytelling. *ACM Transactions on Interactive Intelligent Systems*, 2(3):16:1–16:35.

Pradyumna Tambwekar, Murtaza Dhuliawala, Animesh Mehta, Lara J. Martin, Brent Harrison, and Mark O. Riedl. 2019. Controllable Neural Story Plot Generation via Reward Shaping. *arXiv*, page 7.

Scott R Turner and Michael George Dyer. 1986. *Thematic knowledge, episodic memory and analogy in MINSTREL, a story invention system*. University of California, Computer Science Department.

Stephen Ware and R. Michael Young. 2011. Cpocl: A narrative planner supporting conflict. In *Proceedings of the 7th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*.

Ronald J. Williams. 1992. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning*, 8(3):229–256.

Sam Wiseman, Stuart M. Shieber, and Alexander M. Rush. 2017. Challenges in data-to-document generation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Lili Yao, Nanyun Peng, Ralph Weischedel, Kevin Knight, Dongyan Zhao, and Rui Yan. 2019. Plan-And-Write: Towards Better Automatic Storytelling. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19)*.