Learning to Generate Semantic Overlap from Sentence Pairs through Set-Theory Inspired Reinforcement Rewards

Anonymous ACL submission

Abstract

Learning to generate *Semantic Overlap* from a given sentence pair is a challenging task, especially when a large number of supervised training examples are not available to fine-tune a seq-to-seq model. Other challenges related to this task include deciding whether two sentences indeed have some semantic overlap and if yes, how the overlapping information can be conveyed meaningfully while discarding other information. In this paper, we present a novel deep reinforcement learning network called Deep-STIRR, which can be trained to generate such semantic overlap in an unsupervised fashion without any explicitly labeled training data. To achieve this, we propose novel reward functions for semantic overlap inspired by classical set theory's properties. For evaluation, we manually annotated 900 pairs of sentences and experimented with multiple baselines. Our results show that the proposed STIRR method significantly outperforms multiple state-of-the-art baseline methods.

1 Introduction

Semantic Overlap Generation (SOG) from a given sentence pair is a novel and relatively under-explored multi-sequence-to-sequence generation task. The goal of *SOG* task is to convey the semantic overlap between two or more sentences in a natural way and, at the same time, avoid other information that is not common between the input sentences. *SOG* task can be very useful and serve as a building block for other downstream NLP tasks like Text Summarization, Intelligence Gathering, Sentence Compression, Question Answering (Muralikrishna and Reddy, 2015), etc. Figure 1 shows a toy example of the *SOG* task.

In general, *SOG* is a challenging task, specially when a large number of supervised training examples are not available to fine-tune a seq-to-seq model. There are additional challenges as well, for example: 1) How can we decide whether two

S_1	***It failed again a few years later*** *but soon reopened again.*
S_2	*Though its owners worked hard,* ***after several years, it failed.***
$\cap_O$	*It failed after several years.*

Figure 1: An example of SOG task. The common information in the input sentences are shown in bold.

sentences indeed have some semantic overlap or not? 2) In case there is some overlap, how can we convey the common information in a meaningful way while discarding other information? 3) How can we evaluate the "goodness" of a semantic overlap generator? All these are interesting yet open questions that have motivated this study.

The *SOG* task is closely related to the paraphrasing task (Li et al., 2018) with 2 particular differences: 1) *SOG* task takes two or more sentences as input, while the paraphrasing task requires only one sentence as input. and 2) *SOG* task must maintain the constraint of "overlap" criteria, i.e., it should only convey the common information present in multiple sentences. The paraphrasing task does not impose such a commonality constraint.

Encoder-decoder based deep neural networks have recently gained a lot of attraction, specially for sequence-to-sequence generation tasks, (Rush et al., 2015; Chopra et al., 2016; Zhou et al., 2017; Paulus et al., 2017). However, one of the major challenges associated with implementing a seq-to-seq model which can perform the *SOG* task is the lack of readily available training data for supervised learning. One may manually create a training corpus for a particular domain (e.g., news, health, etc.) by spending a significant amount of time and money, yet it is unclear how much it will generalize to other domains. Therefore, an unsupervised approach is desired to address this problem.

Deep Reinforcement Learning (DRL) (Sutton et al., 1998) has proven to perform well in many

NLP tasks including Paraphrasing (Li et al., 2018), Summarization (Laban et al., 2020), Machine Translation (Wu et al., 2018), Dialogue Generation (Li et al., 2016), etc. One particular benefit of DRL methods is their minimal dependence/no dependence on the availability of large amounts of training data, which directly aligns with our goal for designing a SOG generator. Therefore, we adopted a Deep Reinforcement Learning (DRL) approach for designing and training a SOG generator, which does not require any manually annotated training examples.

Unfortunately, the design and implementation of a DRL approach have their own challenges. Firstly, the action space of the DRL model for *SOG* task will be the size of the vocabulary, which is very large. So, it is very challenging to perform well from a random exploration policy at the beginning. To address this challenge, we created a large number of artificial labels automatically and then pretrained the model with those labels to avoid a *cold start*. Another challenge related to *SOG* task is that we can only calculate how good or bad a generated sentence is and interpret that as a reward only after the whole sentence has been generated. But, if there is a small error at the beginning of the generation, it will accumulate during the whole generation process. Besides, we can not just perform *SOG* on a random pair of sentences. There has to be a substantial semantic similarity between the two sentences.

To address the challenges mentioned above, we designed several reward functions which are inspired by classical set theory to determine how good a generated "Overlap Sentence" is. Specifically, we leveraged existing word similarity metrics to design the reward functions. Still, designing such reward functions is challenging because "goodness" of a "Overlap Sentence" is indeed a philosophical question and there is no single correct answer. In this work, we adopted two simple rules while computing our reward functions:

1. If the generated sentence, Y contains any information that is not common in both input sentences, penalize Y.
2. If Y contains any information that is present in both input sentences, reward Y.

In summary, we propose a deep reinforcement learning approach for generating semantic overlap from sentence pairs which is guided by our novel **S**et-**T**heory **I**nspired **R**einforcement **R**ewards; we call it Deep-STIRR, which can be trained in an unsupervised fashion without any explicitly labeled training data. For evaluation, we manually annotated 900 pairs of sentences by engaging three human volunteers. We then experimented with multiple baselines including word deletion and vanilla paraphrasing. Experimental results show that our STIRR method significantly outperforms multiple state-of-the-art baseline methods, demonstrating the feasibility and utility of our proposed idea.

2 Related Works

Semantic Overlap Generation (SOG) from a given sentence pair is a relatively under-explored task. However, *SOG* task is closely related to the Sentence Fusion task (Ben-David et al., 2020), which has been studied extensively in the literature in different contexts, e.g., abstractive summarization, merging of overlapping sentences in a multi-document context (Filippova and Strube, 2008), combining sentence pairs in a single document (Elsner and Santhanam, 2011), discourse-based sentence fusion (Geva et al., 2019), etc. From the set theory perspective, Sentence Fusion can be viewed as the "union" of multiple sentences, where, the goal is to generate a single sentence that fuses all the information provided by multiple input sentences.

SOG task can be viewed as a special type of Sentence Fusion, where multiple semantically similar sentences are fused together to generate a single sentence while retaining the common information present in the input sentences. From a set theory perspective, it can be viewed as the "intersection" of multiple sentences. Barzilay et al. (1999); Barzilay and McKeown (2005) first introduced the task of sentence intersection as a type of sentence fusion where strictly the shared information among the inputs will be present in the context of multi-document summarizing. The set-like paradigm for sentence intersection was introduced by Marsi and Krahmer (2005) along with other sentence fusion tasks like sentence union.

Filippova and Strube (2008) presented an unsupervised sentence fusion method based on dependency structure alignment and phrase aggregation and pruning. They took a group of related sentences, aligned their dependency trees, and built a dependency graph. Then they used integer linear programming for compressing the graph to a new tree which then they linearized which gives the fi-

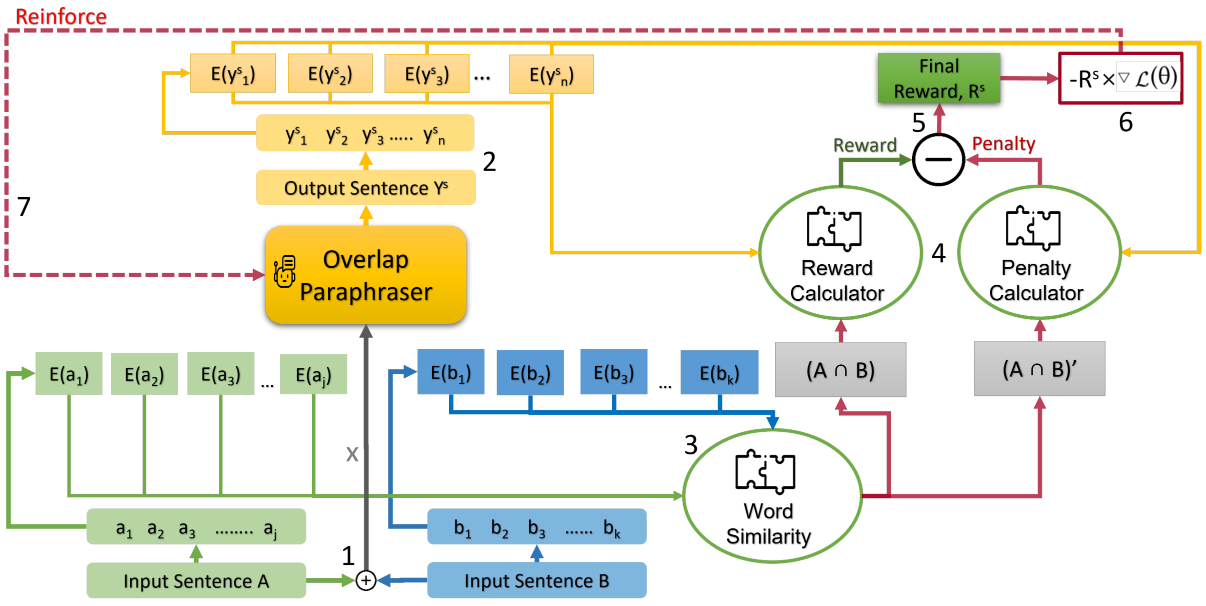

Figure 2: Deep-STIRR approach for semantic overlap generation.

nal output. Thadani and McKeown (2011) used a phrase-based alignment approach to align the input sentences and then selected a subset of the aligned components to generate the overlap/intersection with the help of a dependency parser and beam search. Levy et al. (2016) explored extractive sentence overlap via subtree entailment. They suggested sentence overlap of a pair of similar sentences should be a set of output sentences instead of only one. Besides as their method is extractive, they introduced the idea of 'placeholders' for words that need to be at the generated overlap/intersection but are not present in any of the input sentences.

Still, the task of sentence overlap generation has been under-studied, specially in the context of applying deep learning, mainly due to the lack of annotated data. In this work, we focus on this problem and develop an unsupervised method via deep reinforcement learning to train a semantic overlap generator given pair of sentences as inputs.

3 Proposed Deep-STIRR Approach

In this section, we first provide a high-level overview of our proposed Deep-STIRR approach and then, describe each step in more detail later.

3.1 Overview

An overview of the Deep-STIRR approach is visually depicted in Figure 2. It consists of the following components:

1. **Agent**: An encoder-decoder architecture based Paraphraser model serves as the agent. It takes the concatenated source sentences as input X and encodes it. Then it generates the output sentence Y' word by word at each time-step i. The agent keeps generating until the "End of Sentence" token is generated or the maximum number of token generation limit is reached.
2. **Action-Space**: At a time-step, the agent takes an action $y'_i \in V$, where V is the possible action space, i.e., the vocabulary for generation. After the model is done with generation, the action sequence $Y' = (y'_1, y'_2, ..., y'_m)$ is completed.
3. **Policy**: Action at a time-step i is taken according to the policy $P(y'_i|y'_{1:i-1}, X)$. The state includes the hidden states and the previous actions(i.e.outputs) of the decoder.
4. **Reward/Penalty Calculator**: The final reward R is calculated by analyzing the quality of the generated output, which is computed with the help of a word-similarity module, reward calculator, and penalty calculator. Then we try to optimize the model for maximizing R.

3.2 Design of the Reward Function

We propose novel reward functions which basically calculate how "good" the generated sentences are in conveying the actual overlap (if any) between two sentences. Here, without loss of generality, we assume two sentences as inputs which can be trivially extended to multiple sentences. But, how can we calculate rewards for overlap output?

If we had ground truth labels available, we could calculate the BLEU (Papineni et al., 2002) or ROUGE (Lin, 2004) scores and interpret them as rewards. But as we do not have the labels, we formulated our own reward functions. Our reward functions leverage the power of state-of-the-art word embeddings. In fact, we created multiple variants of the reward function which are similar but have subtle differences and tested those variants of the reward function separately. An overview of the reward calculation procedure is also depicted in figure 2, which combines three separate submodules: word-similarity module, reward calculator, and penalty calculator. The different variants of the reward function are discussed below.

3.2.1 Reward Functions With Threshold

In the first step, the two input sentences and the output sentence are broken into words, and the stopwords are removed from them. Then the words are converted into their embedding vector representation. Next, we compare the cosine similarity of each word from input sentence A with each word of input sentence B. If the cosine similarity of two vectors crosses a user-defined threshold, we consider them as a semantically similar word pair. The similar words in sentences A and B form the set $(A \cap B)$. The words of sentence A or B, which do not match with any words in their counterpart, form the set $(A \cap B)'$. Next, we match each word from the generated sentence Y with $(A \cap B)$ and $(A \cap B)'$ to get $(Y \cap (A \cap B))$ and $(Y \cap (A \cap B)')$, respectively. More elements in set $(Y \cap (A \cap B))$ means a "good" overlap output and more elements in $(Y \cap (A \cap B)')$ means a "bad" output. We have designed two alternative reward functions based on this intuition as shown below.

$$R_1 = |(Y \cap (A \cap B))| \times \frac{\lambda}{|Y|} - |(Y \cap (A \cap B)')| \times \frac{(1-\lambda)}{|Y|} \tag{1}$$

$$R_2 = |(Y \cap (A \cap B))| \frac{\lambda}{|Y|} - (1-\lambda) \left[1 - \frac{|(Y \cap (A \cap B)')|}{|Y|} \right] \tag{2}$$

3.2.2 Reward Function Without Threshold

We also designed a reward function that does not explicitly require a user-defined threshold value. Steps 1 and 2 are the same as the above method. In step 3, we generate three lists after calculating the similarity of the words in sentences A and B. The lists are $(A \cap B)$, $(A - B)$, and $(B - A)$. In step 4, we compare each word (y_p) from the generated sentence Y with all the words in those three lists with respect to their cosine similarity. For each (y_p), we calculate three numbers m_1, m_2, and m_3. More specifically, we compute m_1 by comparing each y_p against all the words in $(A \cap B)$ and finding the maximum cosine similarity score. For finding the maximum cosine similarity score, we don't need any threshold. Similarly, m_2 and m_3 are calculated from $(A - B)$, and $(B - A)$. Then for each word y_p, a score s_p is calculated by subtracting m_2 and m_3 from m_1. Mathematically, the reward is calculated as follows.

$$R_3 = \frac{\sum_{p=0}^{|Y|} s_p}{|Y|}; \quad \text{where, } s_p = m_{1p} - m_{2p} - m_{3p} \tag{3}$$

3.2.3 Fluency Reward

If we ask our model to only optimize on *Overlap* reward, it will not have any incentive to write good fluent English. Generating a sentence that yields the best *Overlap* reward while not caring about the grammar would not be ideal. The purpose of the fluency reward is to maintain the fluency of the generated text independent of the quality of the *Overlap*. We choose the (Laban et al., 2020) implementation of generative Transformer (Radford et al., 2019a) architecture as our fluency model.

3.2.4 Final Reward Function

The final reward is calculated as the weighted sum of the *Overlap* and *Fluency* rewards as follows, where α is a user-defined hyper-parameter.

$$R = \alpha \times OverlapR + (1 - \alpha) \times FluencyR \tag{4}$$

4 Unsupervised Training of Deep-STIRR

We have developed two alternative methods for reinforcement learning in this work. The difference between them is one of the training procedures does not involve a self-critic algorithm and the other procedure does. The first one is called Deep-STIRR, which follows the classic REINFORCE approach (Williams, 1992). And the second one is called Self-critic Deep-STIRR, which we adopt from the Self-critical sequence training(SCST) (Rennie et al., 2016) framework.

4.1 Deep-STIRR Training (REINFORCE)

Figure 2 depicts REINFORCE. The goal of REINFORCE is to optimize the model in such a way that

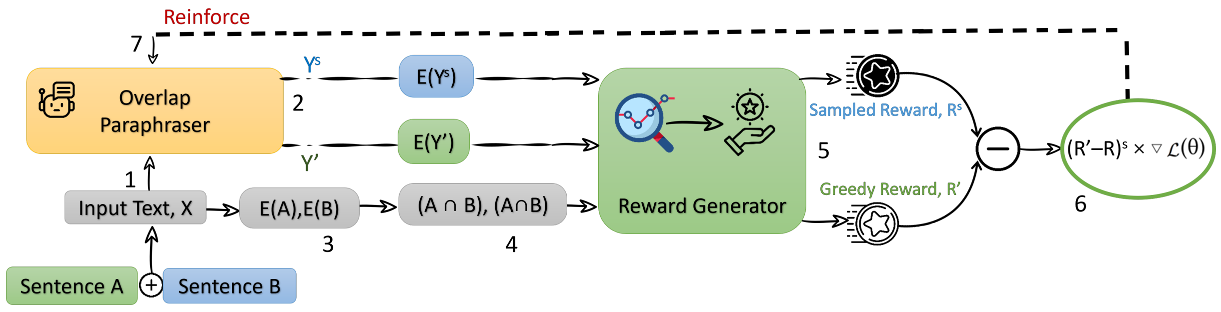

Figure 3: SCST paradigm for unsupervised intersection generation. E(w) stands for embedding of a word w.

will maximize the expected reward R^s for the generated action sequence $Y^s = (y_1^s, y_2^s, ..., y_n^s)$. We are using the sampling method to predict the next word y_i^s from the logits generated by the model during training. The loss for a generated action sequence is:

$$\mathcal{L} = -R^s \times \sum_{i=0}^{N} \log p(y_i^s | y_1^s, ..., y_{i-1}^s, X) \quad (5)$$

Where $p(y_i^s|...)$ represents the probability of the i-th word depending on the generated words so far, according to the model. R^s represents the reward calculated from the generated text. In other words, the loss is the negative expected reward for the generated sequence. This is an approximation as in (Williams, 1992), because there is an infinite number of possible samples that make the calculation infeasible.

REINFORCE Steps (marked in figure 2:)

1. Input sentences A and B are concatenated and passed to the Paraphraser agent.
2. The agent generates output Y^s and passes it to the reward calculator.
3. A and B are tokenized and their word embeddings are passed to the word similarity module.
4. The Similarity module separates $(A \cap B)$ and $(A \cap B)'$ and passes them to reward calculator.
5. The reward calculator calculates the reward R^s for the generated sentence Y^s.
6. Then the loss is calculated using equation(5).
7. The loss is back-propagated and the model parameters are updated.

4.2 Self-Critic Deep-STIRR Training (SCST)

We also tried Self-critical Sequence Training method as it has shown good performance in other NLP tasks such as unsupervised abstractive summarization (Laban et al., 2020). SCST is an extension of REINFORCE with a baseline correction.

The policy gradient generated by REINFORCE can be generalized to compute the reward of an action relative to a reference reward, which we will call baseline, b. One constraint on b is that it can't depend on the action y_i^s. This baseline significantly reduces the variance of the estimated gradient and it also does not change the expected gradient (Rennie et al., 2016). The intuition behind this is that samples from the model that return higher rewards than the baseline will be 'pushed up' or increased in probability. On the other hand, samples with lower rewards will be 'pushed down' in probability.

In SCST, the agent produces two outputs instead of one, unlike in REINFORCE. Two outputs generated are greedy(argmax) overlap Y' and sampled overlap Y^s. The greedy overlap is generated by using a decoding method that always selects the most likely word. And for sampled overlap, the next word is selected by sampling from the word distribution. Then rewards for generated greedy and sampled overlap R' and R^s are calculated. In SCST, we will baseline the REINFORCE algorithm with R'. The loss we need to optimize in this scenario is:

$$\mathcal{L} = (R' - R^s) \sum_{i=0}^{N} \log p(y_i^s | y_1^s, ..., y_{i-1}^s, X) \quad (6)$$

SCST Steps:(As marked in the figure 3)

1. Step 1 to 4 are the same as REINFORCE steps. In step 2, instead of only Y^s, Y' is also generated and processed similarly.
5. The reward calculator calculate R^s and R' for the generated sentences Y^s and Y'.
6. Then the loss is calculated using equation (6).
7. The loss is back-propagated and the model parameters are updated.

5 Experimental Design

5.1 Dataset

We need pairs of overlapping sentences to find the actual overlap between them. To the best of our knowledge, there is no large dataset for doing supervised learning on this task. There have been previous efforts for creating a sentence overlap datasets, such as (McKeown et al., 2010), (Thadani and McKeown, 2013), and (Levy et al., 2016). But due to the complexity of annotation, there has not been any large enough annotated dataset. We also tried to collect the above-mentioned datasets, but the provided links for the datasets in those papers do not work anymore.

There are several paraphrase datasets that contain pairs of similar sentences and labels for if the sentence pair is paraphrases of each other or not. Such a dataset is PAWS (Zhang et al., 2019c). This dataset will provide us with the input sentence pairs we need, but not the ground truth for rigorous evaluation. Therefore, for testing purposes, we annotated 900 sentence pairs. Among those 900, 483 samples had some overlap (according to human judgement) and the other pairs were not similar enough to consider overlap. The histogram of the token lengths of these 966 sentences from those 483 sentence pairs is reported in figure 4. We used these 483 samples for reporting the ROUGE, BERTScore (Zhang et al., 2019b), SARI (Xu et al., 2016), and SEM-F1 (Bansal et al., 2022) scores. Among these 483 samples, we used 150 samples as the validation set and the rest as the test set.

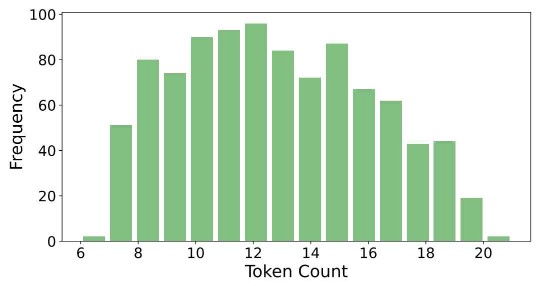

Figure 4: Histogram of token Counts of sentences with significant semantic overlaps in the annotated dataset.

5.2 Finding Overlapping Sentence Pairs

Our model is expected to generate the semantic overlap between two sentences if there is indeed some overlap. But what happens when the sentence pair does not have enough semantic similarity and hence there is no intersection? For handling sentence pairs that do not have enough semantic similarity, we used sentence embeddings and their similarity scores to see whether a particular pair of sentences has enough similarity to have some kind of overlap. In our experiment, we used Sentence BERT(Reimers and Gurevych, 2019) embeddings for this task on the 900 sentence pairs. The performance of our method is stated in Table 1.

Precision	Recall	F-1 Score	Accuracy
0.793	0.944	0.862	0.838

Table 1: Performance of Sentence BERT on the annotated dataset to determine if a sentence pair has enough similarity to have an intersection or not.

5.3 Baseline Methods

Given that we can identify sentence pairs that may have some degree of overlap with a reasonable accuracy using sentence encodes, we now describe some baseline methods we experimented with for generating the actual overlap.

5.3.1 Pretraining

Base on our initial findings as well as recommendations from these works (Ranzato et al., 2015; Siddique et al., 2020), we realized that training a model from scratch with reinforcement without any pre-training might be problematic. One of the reasons for this is the vocabulary size, which is the action set for the paraphraser agent. In the initial state, it is hard for the model to predict the correct words. And if the model makes a mistake at an initial step, the error will keep building up. So, to save the model from global exploration, we pretrained our model with artificial examples. It is to be noted that, our final model trained with reinforcement algorithms performed significantly better than the pretrained model. The comparison of the performance of the models used for pretraining and the models trained with reinforcement learning can be found in Table 3.

5.3.2 Word Delete and Paraphrase (WDAP)

We came up with a method to generate artificial overlap sentences which we call “Word Deletion and Paraphrase (WDAP)”. We later used these artificially created samples to tackle the cold start problem during reinforcement learning.

This method is fairly simple but has a trick. Let’s say we have two input sentences, sentence S and sentence L. At first, we find the smaller sentence be-

Method		R1	R2	RL	BERT Score	SARI	SEM-F1
SBERT	P	0.69	0.43	0.60	0.60	39.69	0.65
	R	0.51	0.31	0.53	0.46		
	F	0.56	0.34	0.54	0.53		
ELMO	P	0.72	0.45	0.63	0.65	40.42	0.68
	R	0.55	0.35	0.49	0.49		
	F	0.60	0.38	0.53	0.59		
GLOVE	P	**0.75**	**0.54**	**0.67**	**0.66**	**45.59**	**0.75**
	R	**0.69**	**0.48**	**0.61**	**0.57**		
	F	**0.70**	**0.49**	**0.62**	**0.61**		

Table 2: The ROUGE-1, ROUGE-2, ROUGE-L, BERTScore, SARI, and average SEM-F1 scores obtained in the validation dataset using WDAP with various word embeddings. P, R, and F stands for Precision, Recall, and F-measure. The best scores are shown in bold.

tween S and L. The reason we take the smaller sentence is that the smaller sentence should be more similar to the expected overlap compared to the larger sentence. Let's say the smaller sentence is S. Now we delete all the words from sentence S which are not semantically similar "enough" (using cosine similarity) to any of the words in sentence L. Then we pass the remaining words in sentence S to the paraphraser model to generate the overlap. In set theory terminology, we find $S - (S \cap L)'$ and pass it to a paraphrasing model to generate the overlap. For paraphrasing, we used a PEGASUS (Zhang et al., 2019a) model fine-tuned for paraphrasing[1]. We have used Huggingface Transformers (Wolf et al., 2020) implementation of PEGASUS.

5.3.3 Embedding Variations

We have tried three methods for word similarity measurement. These methods include generating word embeddings and then measuring similarity between a pair of embeddings with the help of cosine similarity. The three different embedding generation methods we used are:

1. GLOVE (Global Vectors for Words Representations) (Pennington et al., 2014)
2. SBERT (Sentence Embeddings using Siamese BERT-Networks.) (Reimers and Gurevych, 2019)
3. ELMO (Embeddings from Language Models)(Peters et al., 2018)

[1]`https://huggingface.co/tuner007/pegasus_paraphrase`

5.3.4 Fine-tuned End-to-End Model (FM)

We warm-started our reinforcement learning by finetuning the model with artificially generated labels. This pre-training saved our model from expensive global exploration during the initial phase of training. We trained two models with the same reinforcement learning algorithm, but one of the models was finetuned on the artificially generated labels from WDAP. We found the finetuned model performed significantly better than the model without any finetuning (see table 4). We tried several models to fine-tune which include GPT2 (Radford et al., 2019b), PEGASUS, and the transformer encoder-decoder model proposed by (Vaswani et al., 2017).

The PEGASUS model showed much better performance during fine-tuning compared to the other models. Therefore, we chose the PEGASUS model as our final Paraphraser agent.

Method		R1	R2	RL	BERT Score	SARI	SEM-F1
JP	P	0.67	0.50	0.61	0.63	46.26	0.74
	R	0.77	0.59	0.70	0.64		
	F	0.68	0.52	0.64	0.63		
WDAP	P	0.75	0.52	0.66	0.68	44.89	0.77
	R	0.69	0.48	0.61	0.61		
	F	0.70	0.48	0.62	0.65		
Finetuned Model	P	0.71	0.50	0.63	0.65	44.89	0.74
	R	0.71	0.49	0.63	0.60		
	F	0.68	0.48	0.61	0.62		
RE	P	0.85	0.71	0.80	0.73	57.71	0.85
	R	0.84	0.70	0.79	0.70		
	F	0.83	0.69	0.78	0.72		
SCST(R_3)	P	0.82	0.69	0.79	0.72	56.92	0.82
	R	0.83	0.70	0.80	0.68		
	F	0.81	0.68	0.78	0.70		
SCST(R_2)	P	**0.87**	**0.73**	**0.82**	**0.74**	57.04	0.85
	R	0.82	0.69	0.78	0.69		
	F	0.83	0.69	0.79	0.72		
SCST(R_1)	P	0.86	0.73	0.82	0.74	**58.08**	**0.85**
	R	**0.85**	**0.71**	**0.81**	**0.71**		
	F	**0.84**	**0.71**	**0.80**	**0.72**		

Table 3: ROUGE, BERTScore, SARI, and average SEM-F1 Scores obtained using various methods on Test dataset. The best scores are shown in bold.

6 Results

We first report our findings on using different embedding variants. Using GLOVE embeddings to find the semantic similarity among words gave the best ROUGE, BERTScore, SARI, and SEM-F1 scores in the validation dataset for our task. The scores are reported in table 2. Therefore, we chose to use GLOVE embeddings for the rest of our methods in this work.

Table 3 reports the performance of the different methods we propose in this work. We also report the scores of a simple method Just Paraphrase (JP) as a baseline. In this method, we concatenated the input sentences and passed them through a PEGASUS paraphrase model to get the output. Note that, our reinforcement algorithm starts with the Finetuned Model (FM) and the Finetuned Model is trained with the artificial samples generated by the WDAP method; therefore, we consider these two methods as baselines as well. Table 3 clearly shows that the models trained with reinforcement learning algorithms and our reward function performed better than the baseline methods in all the metrics.

While training the models, after a certain number of iterations, the rewards were increasing but the actual validation ROUGE, SARI, and other scores were dropping. This is because the reward functions we proposed have limitations. After a certain amount of iterations, the models trick the reward function to maximize reward but the quality of the generated output drops substantially. So, we took the models with the best validation scores and reported the test scores for them. The reward and loss curves of the proposed methods can be found in the appendix.

6.1 Ablation Study

We performed a detailed ablation study for our proposed approach. Table 4 shows test scores for three models. For the model Without Pretraining, we started self-critical sequence training with a Pegasus paraphraser model without pretraining it first with our artificially created dataset. And for training the model Without Fluency, we did not include any fluency component in the reward as in equation 4. And the final model is our proposed model. We can clearly see that adding pretraining helps the model gain better performance. And if we look at the scores for the model Without Fluency, we find that it beats the proposed model in Rouge R1, R2, R3, and BERTScore precision scores. But the proposed model beats that model in all the other metrics. This is because the Without Fluency model tries to maximize the overlap reward without caring about fluency. This results in poorer overall performance. Figure 5 shows the rewards obtained during training of the methods mentioned above.

Method		R1	R2	RL	BERT Score	SARI	SEM-F1
Without Fluency	P	**0.89**	**0.74**	**0.83**	**0.75**	56.85	0.85
	R	0.81	0.68	0.77	0.68		
	F	0.83	0.69	0.78	0.72		
Without Pretraining	P	0.71	0.56	0.66	0.65	51.05	0.79
	R	0.82	0.65	0.75	0.69		
	F	0.75	0.59	0.69	0.67		
SCST(R_1)	P	0.86	0.73	0.82	0.74	**58.08**	**0.85**
	R	**0.85**	**0.71**	**0.81**	**0.71**		
	F	**0.84**	**0.71**	**0.80**	**0.72**		

Table 4: ROUGE, BERTScore, SARI, and average SEM-F1 Scores obtained during ablation analysis on Test dataset.

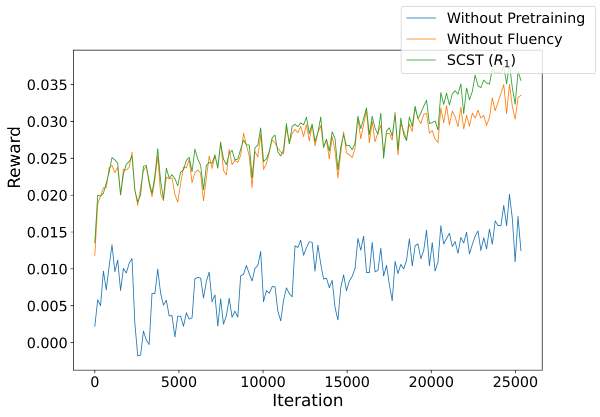

Figure 5: Training Overlap Rewards obtained during ablation analysis.

7 Conclusion and Future Work

In this work, we introduce a new approach to generating semantic overlap for a given sentence pair based on maximizing the information commonality within the generated sentence. The major benefit of our proposed approach is that it does not require manually labeled training data. We harnessed the power of SOTA Deep Reinforcement Learning Techniques and performed exhaustive experiments to find the best reward functions and training procedures for this task. We also annotated 900 sentence pairs for this task and showed that our proposed method performed significantly better than the reported baselines. In future works, we also plan to investigate the effectiveness of our proposed approach for long documents.

References

Naman Bansal, Mousumi Akter, and Shubhra Kanti Karmaker Santu. 2022. Multi-narrative semantic overlap task: Evaluation and benchmark.

Regina Barzilay and Kathleen R. McKeown. 2005. Sentence fusion for multidocument news summarization. *Computational Linguistics*, 31(3):297–328.

Regina Barzilay, Kathleen R. McKeown, and Michael Elhadad. 1999. Information fusion in the context of multi-document summarization. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pages 550–557, College Park, Maryland, USA. Association for Computational Linguistics.

Eyal Ben-David, Orgad Keller, Eric Malmi, Idan Szpektor, and Roi Reichart. 2020. Semantically driven sentence fusion: Modeling and evaluation. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1491–1505.

Sumit Chopra, Michael Auli, and Alexander M Rush. 2016. Abstractive sentence summarization with attentive recurrent neural networks. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 93–98.

Micha Elsner and Deepak Santhanam. 2011. Learning to fuse disparate sentences. In *Proceedings of the Workshop on Monolingual Text-To-Text Generation*, pages 54–63, Portland, Oregon. Association for Computational Linguistics.

Katja Filippova and Michael Strube. 2008. Sentence fusion via dependency graph compression. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 177–185, Honolulu, Hawaii. Association for Computational Linguistics.

Mor Geva, Eric Malmi, Idan Szpektor, and Jonathan Berant. 2019. Discofuse: A large-scale dataset for discourse-based sentence fusion. *CoRR*, abs/1902.10526.

Philippe Laban, Andrew Hsi, John Canny, and Marti A. Hearst. 2020. The summary loop: Learning to write abstractive summaries without examples. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics.

Omer Levy, Ido Dagan, Gabriel Stanovsky, Judith Eckle-Kohler, and Iryna Gurevych. 2016. Modeling extractive sentence intersection via subtree entailment. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 2891–2901, Osaka, Japan. The COLING 2016 Organizing Committee.

Jiwei Li, Will Monroe, Alan Ritter, Michel Galley, Jianfeng Gao, and Dan Jurafsky. 2016. Deep reinforcement learning for dialogue generation.

Zichao Li, Xin Jiang, Lifeng Shang, and Hang Li. 2018. Paraphrase generation with deep reinforcement learning. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3865–3878, Brussels, Belgium. Association for Computational Linguistics.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

Erwin Marsi and Emiel Krahmer. 2005. Explorations in sentence fusion. In *Proceedings of the Tenth European Workshop on Natural Language Generation (ENLG-05)*, Aberdeen, Scotland. Association for Computational Linguistics.

Kathleen McKeown, Sara Rosenthal, Kapil Thadani, and Coleman Moore. 2010. Time-efficient creation of an accurate sentence fusion corpus. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 317–320, Los Angeles, California. Association for Computational Linguistics.

RVV Muralikrishna and Ch Satyananda Reddy. 2015. A survey on sentence fusion techniques of abstractive text summarization. *International Journal of Engineering Research and Applications*, 5(5):59–64.

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 of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Romain Paulus, Caiming Xiong, and Richard Socher. 2017. A deep reinforced model for abstractive summarization. *arXiv preprint arXiv:1705.04304*.

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, Doha, Qatar. Association for Computational Linguistics.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana. Association for Computational Linguistics.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019a. Language models are unsupervised multitask learners.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019b. Language models are unsupervised multitask learners.

Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. 2015. Sequence level training with recurrent neural networks.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Steven J. Rennie, Etienne Marcheret, Youssef Mroueh, Jarret Ross, and Vaibhava Goel. 2016. Self-critical sequence training for image captioning.

Alexander M Rush, Sumit Chopra, and Jason Weston. 2015. A neural attention model for abstractive sentence summarization. *arXiv preprint arXiv:1509.00685*.

A. B. Siddique, Samet Oymak, and Vagelis Hristidis. 2020. Unsupervised paraphrasing via deep reinforcement learning. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, page 1800–1809, New York, NY, USA. Association for Computing Machinery.

Richard S Sutton, Andrew G Barto, et al. 1998. Introduction to reinforcement learning.

Kapil Thadani and Kathleen McKeown. 2011. Towards strict sentence intersection: Decoding and evaluation strategies. In *Proceedings of the Workshop on Monolingual Text-To-Text Generation*, pages 43–53, Portland, Oregon. Association for Computational Linguistics.

Kapil Thadani and Kathleen McKeown. 2013. Supervised sentence fusion with single-stage inference. In *Proceedings of the Sixth International Joint Conference on Natural Language Processing*, pages 1410–1418, Nagoya, Japan. Asian Federation of Natural Language Processing.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need.

Ronald J. Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Lijun Wu, Fei Tian, Tao Qin, Jianhuang Lai, and Tie-Yan Liu. 2018. A study of reinforcement learning for neural machine translation.

Wei Xu, Courtney Napoles, Ellie Pavlick, Quanze Chen, and Chris Callison-Burch. 2016. Optimizing statistical machine translation for text simplification. *Transactions of the Association for Computational Linguistics*, 4:401–415.

Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter J. Liu. 2019a. Pegasus: Pre-training with extracted gap-sentences for abstractive summarization.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. 2019b. Bertscore: Evaluating text generation with bert.

Yuan Zhang, Jason Baldridge, and Luheng He. 2019c. PAWS: Paraphrase Adversaries from Word Scrambling. In *Proc. of NAACL*.

Qingyu Zhou, Nan Yang, Furu Wei, and Ming Zhou. 2017. Selective encoding for abstractive sentence summarization. *arXiv preprint arXiv:1704.07073*.

A SUPPLEMENTARY MATERIAL

A.1 Limitations

One limitation of our proposed method is that for determining if two words are similar we are using word embeddings which have many limitations, such as not being able to detect two semantically similar words. So, this method is as good as the word similarity measuring method we are using.

Besides, the methods require a decent amount of time for converging. One of the reasons for that is in every iteration, we are required to calculate rewards for the generated sentences. And reward calculating methods are heavily time and memory consuming. Also, PEGASUS itself is a heavy model with 568M parameters. So, we utilized 18 GB of GPU memory for around 10 hours to train our models with a batch size of 8.

The notion of sentence intersection is complex and subjective. The reward calculating functions we defined are fairly simple compared to that. So, at a certain point, the rewards keep increasing but the validation scores fall. Adding fluency reward mitigates the problem, but does not prevent it altogether.

A.2 Training Curves

The loss and reward curves for the methods REINFORCE (R_1), SCST (R_1), and SCST (R_3) can be found in figure 6, 7, and 8.

A.3 Hyper-parameter Tuning

We treat λ in (1) as a hyper-parameter and tried multiple values of it to find the best fit. During hyper-parameter tuning, we used the validation part of the annotated dataset for validation. The validation scores are presented in table 5. And the training algorithm used was SCST(R_1).

Both of the reward calculation functions depend on GLOVE embeddings for word similarity measurement. If the cosine similarity score of two word-embeddings is higher than a threshold value, θ we consider them similar. We also consider this θ as a hyper-parameter and tried different values. The validation scores for tuning θ are present in table 6. The training algorithm used was SCST(R_1).

The best performing λ and θ values found were 0.95 and 0.90 respectively.

λ		R1	R2	RL	BERT Score	SARI	SEM-F1
0.80	P	0.77	0.62	0.71	0.68	54.47	0.82
	R	**0.86**	**0.69**	**0.80**	**0.73**		
	F	0.79	0.64	0.73	0.71		
0.85	P	0.79	0.62	0.72	0.69	54.11	0.82
	R	0.86	0.68	0.79	0.73		
	F	0.80	0.63	0.73	0.71		
0.90	P	0.85	0.67	0.77	0.73	55.14	0.84
	R	0.83	0.66	0.76	0.71		
	F	0.82	0.64	0.75	0.72		
0.95	P	**0.87**	**0.70**	**0.81**	**0.75**	**56.98**	**0.84**
	R	0.81	0.66	0.77	0.71		
	F	**0.82**	**0.66**	**0.77**	**0.72**		
0.975	P	0.84	0.66	0.76	0.73	55.0	0.84
	R	0.83	0.66	0.76	0.71		
	F	0.82	0.64	0.74	0.72		

Table 5: Validation scores obtained during tuning of λ.

θ		R1	R2	RL	BERT Score	SARI	SEM-F1
0.80	P	0.83	0.66	0.76	0.73	55.52	0.83
	R	**0.83**	0.67	0.77	**0.72**		
	F	0.81	0.64	0.75	0.72		
0.85	P	0.87	0.70	0.81	0.75	**56.98**	0.84
	R	0.81	0.66	0.77	0.71		
	F	0.82	0.66	0.77	0.72		
0.90	P	**0.88**	**0.71**	**0.81**	**0.76**	56.76	**0.84**
	R	0.81	0.66	0.75	0.71		
	F	**0.83**	**0.67**	**0.77**	**0.73**		
0.95	P	0.83	0.68	0.78	0.72	55.73	0.84
	R	0.83	**0.69**	**0.78**	0.69		
	F	0.82	0.67	0.77	0.70		

Table 6: Validation scores obtained during tuning of θ.

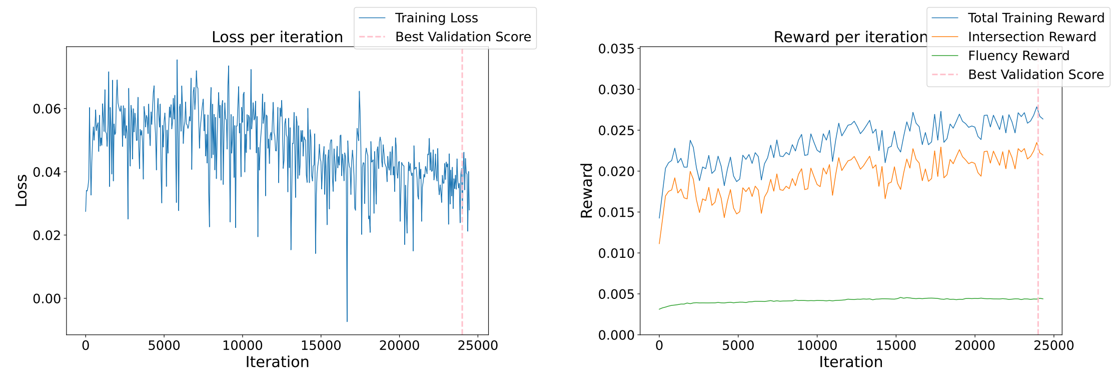

Figure 6: Training Loss and Reward curves for method REINFORCE (R_1).

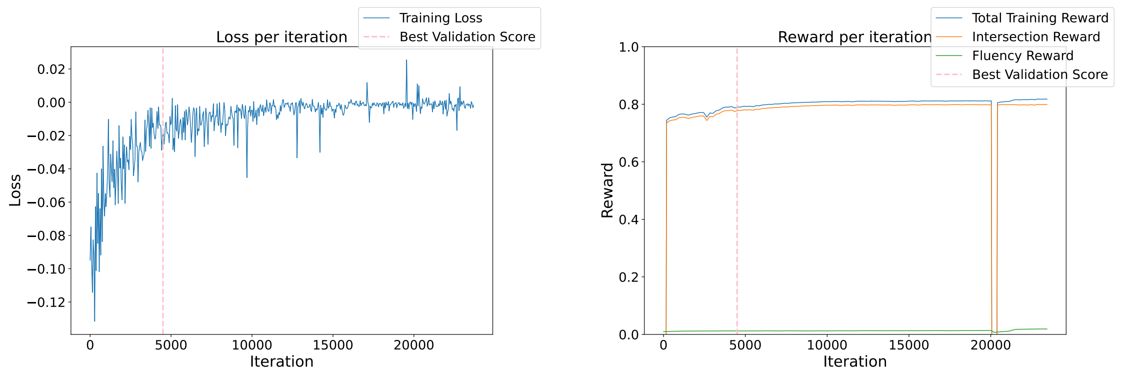

Figure 7: Training Loss and Reward curves for method SCST (R_3).

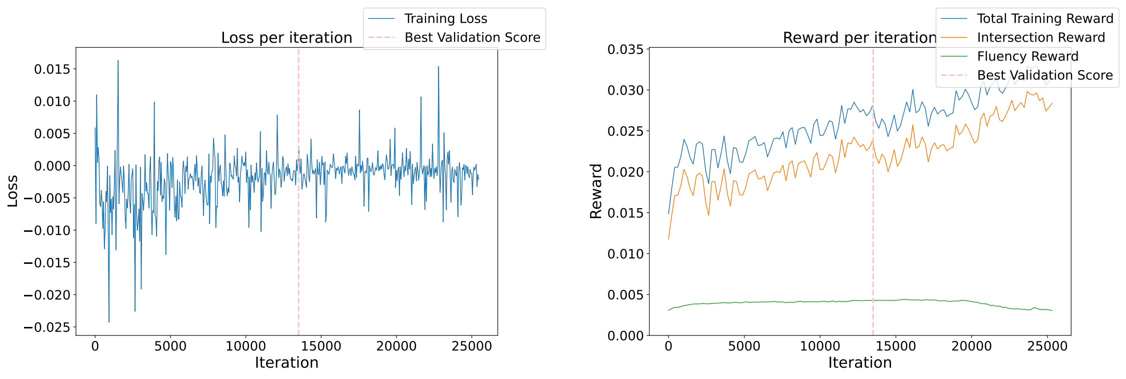

Figure 8: Training Loss and Reward curves for method SCST (R_1).