The Text-Minder

Designing a Framework for Controlled Language Generation Through the Use of Interpretable Semantic Constraints

Jerry Ding
Machine Learning Department
Carnegie Mellon University
Pittsburgh, PA 15213
dalud@andrew.cmu.edu

Code and results are available on: https://github.com/JubilantJerry/text-minder

1 Introduction and Overall Goal

The success of neural models such as BERT [1] on wide classes of natural language processing tasks, sometimes even above what specialized tools designed by experts can achieve, have led researchers to wonder whether a single neural model can be used to capture nearly all linguistic knowledge in one domain. This perspective has led to an increased reliance on end-to-end models where one hopes a single neural network would automatically learn to perform syntactic parsing, semantic processing, and domain-specific reasoning when trained for a specific task. Increasingly, people look toward large language models such as GPT-2 [2] to solve such multifaceted problems.

Language models learn the probability distribution of word or character sequences in a data set. Once trained, the model can be used to generate text, compute the likelihoods of existing sentences, or even build classifiers through Bayesian inference. Many NLP problems can be formulated as supervised learning tasks with language models, where a problem instance is encoded as a textual prompt, the ground truth label is encoded as a textual reply to the prompt, and the language model is trained to maximize the probability of generating the correct response to each prompt.

A downside to this formulation is that it is difficult for a researcher to control the output of the model beyond fine-tuning the model on their data set and tweaking the prompt text. In particular, researchers are left in an awkward position where they cannot condition the model output on nontextual data, and even if they already have a reasonably well-performing expert system that can extract domain-specific semantics from the prompt and can decide roughly how the system should respond, they have no way of interfacing the expert system with the language model.

This can be an especially big problem if a task is open-ended, such as the task of modeling conversational dialogue. Language is used in conversation to convey novel information. A speaker can respond to a conversational partner in many ways depending on their knowledge, mood, objectives and more, all of which could change even within a single conversation. Even a perfect language model conditioned on the full conversation history is not guaranteed to predict the true response.

In this paper, I propose the Text-Minder framework ("TMD framework" in short) for controlled response generation in conversational contexts. My goal is to offer a method for decoupling the semantics of a response from its non-semantic properties such as word choice and syntax, so that the user is not forced to rely on a single neural model to make correct judgements on both. In the framework, the client provides the semantics of their desired response in a simplified semantic constraint language (the "SC language"), and a language model attempts to generate natural language that closely adheres to the provided constraints.

2 Approach and Desiderata

The framework consists of the following components:

- A specification for the syntax and structure of a simple semantic constraint language
- An automated tool to translate existing sentences into the semantic constraint language
- A language model that attempts to recreate natural language given several lines of conversational context and the semantic constraints representing the desired output

2.1 Data Set

To create these three components, it is important to have a data set of conversational dialogue to use as realistic examples and as training data for the language model. I choose to use the OpenSubtitles [3] data set, because it contains a large number of conversations under a wide variety of conversational contexts. I believe this data set is a good approximation to casual conversation between friends; one direction for future work could be to augment the TMD framework to better suit more formal or domain-specific conversational language.

2.2 Semantic Constraint Language

This language is the medium through which the TMD framework and its users interact. The success of the design of this language should not be measured by how accurately the TMD language model is able to recover a sentence from its SC representation. Otherwise, a trivial choice for the SC language would be the natural language itself, i.e. the user directly specifies what their desired sentence is, and the framework simply echos the provided sentence. Instead, my goal is to design the language to satisfy the following three desiderata:

1. **Human Interpretibility** - Ultimately, a human researcher or programmer creates the interface between a semantic reasoner and the TMD framework, so they would need to understand the meaning of the semantic constraints in order to properly generate them for the engine.
2. **Isolation** - Each semantic constraint is a meaningful unit of semantic information; the language disallows compound constructs made of individually meaningless constraints, and information should not be encoded in the order of the constraints.
3. **Simple yet Expressive** - It should be easy to produce semantic constraints that roughly encodes the desired response, yet the constraints should encode enough information that outputs from the TMD language model are reasonably close to said desired response.

2.3 Automatic Semantic Constraint Generation

Given an understanding of the SC language, it is possible for human annotators to produce semantic constraints corresponding to any given sentence. However, I have decided to build an automatic semantic constraint generator and use it to create a large number of labeled sentences as training data for the TMD language model. It is expected that the machine generated constraints will contain errors or omissions, but it is hoped that these are rare enough that the constraints are still useful as training data. One possible direction for future work would be to study how big of an impact the errors have on the performance of the TMD language model.

2.4 Constrained Language Model

Using the automatic constraint generator, every line in the OpenSubtitles data set is converted into its SC language representation, and a language model is trained to predict a line of dialogue given up to five lines of context and the semantic constraints for both the context and the desired response. To account for possible differences in how users choose to encode their response as semantic constraints, the constraints in the training data are generated liberally and are randomly dropped out when being presented to the language model during training.

2.5 Evaluation Concerns

It is tricky to decide how to evaluate the effectiveness of the TMD framework, as its three components serve different roles and have very different success criteria.

The SC language is designed for ease of use and the three desiderata listed above. I imagine that the only true test of the language's effectiveness would be through case studies using the TMD framework

in a domain-specific language modeling or text generation task, or with user feedback following public release of a future version of the TMD framework.

The automatic semantic constraint generator aims to create constraints similar to what a human annotator might produce. However, there is some degree of subjectivity in how a given natural language sentence can be translated into the SC language. I imagine that the effectiveness of the constraint generator can be evaluated by first creating a set of human annotated semantic constraints, then conducting a double-blind experiment employing a panel of judges to compare the quality of the hand-written constraints against the automatically generated constraints.

Once the details of the SC language are fixed and the semantic constraints for a data set are generated, the effectiveness of a conditional language model can evaluated relatively objectively. The goal of the language model is to produce natural language sentences from semantic constraints; ideally, if a natural language sentence is translated into semantic constraints via an automatic tool, the language model would be able to recover the original sentence from the constraints, so the quality of the language model can be measured by metrics such as text perplexity or BLEU. One challenge is that it is possible to have more than one valid sentence given the dialogue context and the desired semantics. A possible remedy would be to include multiple reference sentences as labels in the data set.

I believe all three components are important and are worthy of careful evaluation. However, due to resource limitations, in this paper I focus my formal evaluation efforts on the language model, and do not use multiple reference sentences in my evaluation metrics.

3 Work Done

3.1 Designing the SC Language Syntax

Based on visual investigation of the dialogue in the OpenSubtitles data set, and manual attempts to encode sentences into semantic constraints, I choose to encode natural language sentences as multigraphs (graphs with potentially repeated edges) describing the relationships between entities in the sentence. Each entity is given a name in the natural language; these are often just the nouns or pronouns referring to the entity. Semantic constraints are used to encode relationships between two entities, or attach modifiers to a single entity. In this way, the entities form the vertices of the multigraph and the semantic constraints form the edges.

Those who wish to see the full specification of the SC language can refer to a separate document, though I will give a brief summary here. Semantic constraints are made of tokens of three types: entities, keywords, and tags. Tags are basic semantic concepts selected from a small set of 140 items, whereas keywords serve more grammatical purposes such as specifying the type of constraint being used, describing the method in which tags should be interpreted, and introducing placeholders for special types of entities or relationships. The three main constraint types are:

- **Relevant**: Introduces an entity and attaches a number of tags associated with the entity
- **Description**: Attaches modifiers to an entity in the form of tags and keywords
- **Action**: Describes an action or experience, with slots for the agent and the patient. Special relationships between two entities are also encoded with this type of constraint.

In addition, the **Question** and **Utterance** constraints are used to indicate whether a sentence is a question, and whether it contains an interjection. These are not associated with any entity in particular.

3.2 Finding the SC Language Tags

To choose the set of tags used in the SC language, I analyzed the vocabulary of the OpenSubtitles data set to produce an upper ontology of concepts (represented as WordNet [4] synsets). Each of these concepts are then encoded as a set of tags. In addition, I performed k-means clustering on the GLoVe vectors [5] of the vocabulary terms to identify word clusters, and a number of tags was assigned to each of these clusters. In the end, 90 unique tags were used in my annotations of these upper ontology words and word clusters. The top 50 most common verbs were added as an additional 50 tags, which are only used as an encoding of those exact 50 verbs.

3.3 Creating the Tag Generator

To build the semantic constraint generator, a module is needed to generate descriptive tags for arbitrary words in the vocabulary. My implementation generates the tags with the following procedure:

1. If the word is one of the top 50 verbs, then simply return the corresponding tag
2. Sort the upper ontology words based on the graph search distance on the WordNet graph (with hand-tuned edge distances for each word relation type)
3. Extract the tags associated with the closest upper ontology words until a limit is reached in the number of tags or the distance of the upper ontology words
4. Locate the word cluster of the vocabulary term, and add the tags associated with that cluster

In addition, I manually wrote down tags for the top 1000 most frequent English words, and appended these to the automatically generated tags.

3.4 Building the Automatic Semantic Constraint Generator

The constraint generator is implemented to follow the below procedure on a given sentence:

1. Build a parse tree of the sentence with a pre-trained dependency parsing model provided by AllenNLP (which is based on a deep biaffine LSTM network [6]).
2. Use the tag generator module to find tags for the nouns, verbs, adjectives and adverbs in the sentence. These will be used in the semantic constraints.
3. Recursively visit nodes of the parse tree, while generating semantic constraints according to a large list of manual rules. These constraints are accumulated and returned as the output.

3.5 Implementing the TMD Language Model

The TMD framework uses a pair of causal transformer networks similar to GPT-2 as the language model. The first network is the encoder module, which produces embeddings for the SC representations of the context dialogue and the desired response. The second network is the decoder module, which generates each word of the output sentence conditioned on the context dialogue, previously generated words, and the semantic constraint embeddings.

The TMD framework converts the constraints into vectors by concatenating the word embeddings encoding the entity names with bitvectors encoding the keywords and tags. These vectorized constraints are transformed with the encoder module into semantic constraint embeddings, which are passed as input to the decoder module. I include a pretrained GPT-2 345M parameter model as one component of the decoder module, leaving its weights frozen during training. Instead, the hidden layer outputs of the frozen network are projected down to the first 1024 principal components, which are used as input for a number of additional transformer layers (which make up the remainder of the decoder module). This is done to reduce the amount of training time, since GPT-2 is computationally much cheaper to run in inference mode, and the principal components can be cached so that they do not need to be re-computed every epoch.

During training, the order of the semantic constraints is partially randomized, and constraints are dropped out with 25% probability. Only a random subset of the tags from the tag generator are kept; for the top 1000 words, the dropout distribution favors manually written tags over automatically generated tags. On average, three tags are kept and the remainder are dropped. Finally, the transformer networks are regularized with weight decay and dropout to help with generalization performance.

4 Evaluation Results

From the full OpenSubtitles data set, 4.86M sentences are randomly sampled to use as training data. This is also the data used for exploratory data analysis and for computing the word clusters in the tag generator. A separate sample of 1000 sentences is used for evaluation.

The first evaluation metric is the text perplexity of the ground truth response in each problem instance; computing this does not require autoregressive sampling from the language model. Then, for each problem instance I sample 25 sentences from the model, use the automatic constraint generator to generate semantic constraints for these sentences, and pick the sentence whose constraints most

Model Name	Text Perplexity	SC-BLEU	NL-BLEU
Full	3.77	0.955	0.618
Nouns-only	9.06	0.731	0.406
Bare	30.96	0.441	0.165

Table 1: The resulting metrics for the three models.

closely match the SC representation of the ground truth response. Similarity of semantic constraints is measured by tokenizing the constraints and computing the BLEU score. Lastly, I compute the similarity of the chosen sentence against the ground truth, also with the BLEU score. To avoid confusion between these two BLEU scores (which use different tokens and are not comparable), the former is denoted the SC-BLEU score and the latter is denoted the NL-BLEU score. The two BLEU scores are averaged across problem instances in the evaluation set, unlike text perplexity which is averaged across tokens in the ground truth responses. It may seem questionable to select one sample from 25, but this process is fair since the selection process does not require the ground truth response itself, only its SC representation which is one of the givens in the problem.

The three evaluation metrics are computed for three models. The full model is trained exactly as described so far, the nouns-only model is trained with trivial semantic constraints which only list the nouns in a sentence, and the bare model is trained with no semantic constraints. This only affects the input to the language model; the evaluation procedure still computes full SC representations when calculating SC-BLEU and NL-BLEU scores. The results are shown in Table 1.

5 Discussion and Conclusion

The three models do not perform the same task since they do not see the same semantic constraints, so the lower text perplexity and higher BLEU score of the full model is not an argument that the full model is inherently better. However, the large difference in the scores do demonstrate that the semantic constraints are effective in guiding the model toward the ground truth response. The high SC-BLEU value shows that the generated sentences often adhere quite closely to the provided constraints, even if the word choice may be different as indicated by the lower NL-BLEU score.

This set of metrics is still imperfect due to its reliance on an imperfect semantic constraint generator and the lack of a second language model implementation to compare against. In addition, from careful inspection of the generated sentences (available in the evaluation logs of the code repository) one can find sentences containing errors such as having the wrong sentiment polarity, using the wrong prepositions, or placing entities in the wrong semantic roles. This happens even when the SC-BLEU score is high, suggesting that there is room to improve the constraint generator and/or increase the expressiveness of the SC language.

However, I believe that I have demonstrated the plausibility of designing a sensible semantic constraint language and of creating semantic interfaces for conditional language models. I hope that all three components of the TMD framework can be further expanded in future work to create a reliable framework for NLP applications in the industry.

6 Addendum: Uses of semantics

The SC language is itself a semantic representation. The current implementation of the TMD framework uses non-distributional semantic knowledge in the form of WordNet word relationships, the choice of the upper ontology words, manual tagging of common English words, and manual rules for converting parse tree structures into semantic constraints. The framework also uses distributional semantics in the form of GLoVe word vectors and pretrained GPT-2 weights.

References

[1] Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." *arXiv preprint arXiv:1810.04805* (2018).

[2] Radford, Alec, et al. "Language models are unsupervised multitask learners." OpenAI Blog 1.8 (2019): 9.

[3] Lison, Pierre, and Jörg Tiedemann. "Opensubtitles2016: Extracting large parallel corpora from movie and tv subtitles." (2016): 923-929.

[4] Fellbaum, Christiane. "WordNet." *The encyclopedia of applied linguistics* (2012).

[5] Pennington, Jeffrey, Richard Socher, and Christopher D. Manning. "Glove: Global vectors for word representation." *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014.

[6] Dozat, Timothy, and Christopher D. Manning. "Deep biaffine attention for neural dependency parsing." *arXiv preprint arXiv:1611.01734* (2016).