Texar: A Modularized, Versatile, and Extensible Toolbox for Text Generation

Zhiting Hu*, **Zichao Yang**, **Haoran Shi**, **Bowen Tan**, **Tiancheng Zhao**, **Junxian He**, **Xiaodan Liang**, **Wentao Wang**, **Xingjiang Yu**, **Di Wang**, **Lianhui Qin**, **Xuezhe Ma**, **Hector Liu**, **Devendra Singh**, **Wangrong Zhu**, **Eric P. Xing**
zhitingh@cs.cmu.edu*, Carnegie Mellon University, Petuum Inc.

Abstract

We introduce Texar, an open-source toolkit aiming to support the broad set of text generation tasks. Different from many existing toolkits that are specialized for specific applications (e.g., neural machine translation), Texar is designed to be highly flexible and versatile. This is achieved by abstracting the common patterns underlying the diverse tasks and methodologies, creating a library of highly reusable modules and functionalities, and enabling arbitrary model architectures and various algorithmic paradigms. The features make Texar particularly suitable for technique sharing and generalization across different text generation applications. The toolkit emphasizes heavily on extensibility and modularized system design, so that components can be freely plugged in or swapped out. We conduct extensive experiments and case studies to demonstrate the use and advantage of the toolkit.

1 Introduction

Text generation spans a broad set of natural language processing tasks that aim at generating natural language from input data or machine representations. Such tasks include machine translation (Bahdanau et al., 2014; Brown et al., 1990), dialog systems (Williams and Young, 2007; Serban et al., 2016), text summarization (Hovy and Lin, 1998; See et al., 2017), article writing (Wiseman et al., 2017), text paraphrasing and manipulation (Madnani and Dorr, 2010; Hu et al., 2017a), image captioning (Vinyals et al., 2015b; Karpathy and Fei-Fei, 2015), and more. Recent years have seen rapid progress of this active area in both academia and industry, especially with the adoption of modern deep learning approaches in many of the tasks. On the other hand, considerable research efforts are still needed to improve relevant techniques and enable real-world practical applications.

The variety of text generation tasks share many common properties and goals, e.g., to generate well-formed, grammatical and readable text, and to realize in the generation the desired information inferred from inputs. To this end, a few key models and algorithms are increasingly widely-used to empower the different applications, such as neural encoder-decoders (Sutskever et al., 2014), attentions (Bahdanau et al., 2014; Luong et al., 2015b), memory networks (Sukhbaatar et al., 2015), adversarial methods (Goodfellow et al., 2014; Hu et al., 2017b; Lamb et al., 2016), reinforcement learning (Ranzato et al., 2015; Bahdanau et al., 2016), as well as some optimization techniques, data preprocessing and result post-processing procedures, evaluations, etc.

It is therefore highly desirable to have an open-source platform that unifies the development of the diverse yet closely-related applications, backed with clean and consistent implementations of the core algorithms. Such a unified platform enables reuse of common components and functionalities, standardizes design, implementation, and experimentation, fosters reproducible research, and importantly, encourages technique sharing among different text generation tasks, so that an algorithmic advance originally developed for a specific task can quickly be evaluated and potentially generalized to many other tasks.

Though a few remarkable open-source toolkits have been developed, they have been largely designed for one or few specific tasks, especially neural machine translation (Britz et al., 2017; Klein et al., 2017; Neubig et al., 2018) and dialog related algorithms (Miller et al., 2017). This pa-

Proceedings of Workshop for NLP Open Source Software, pages 13–22
Melbourne, Australia, July 20, 2018.

per introduces *Texar*, a general-purpose text generation toolkit that aims to support most of the popular applications in the field, by providing researchers and practitioners a unified and flexible framework for building their models. Texar is built upon TensorFlow[1], a popular deep learning platform. Texar emphasizes on three key properties, namely, versatility, modularity, and extensibility.

- **Versatility**: Texar contains a wide range of features and functionalities for 1) arbitrary model architectures as a combination of encoders, decoders, discriminators, memories, and many other modules; and 2) different modeling and learning paradigms such as sequence-to-sequence, probabilistic models, adversarial methods, and reinforcement learning. Based on these, both workhorse and cutting-edge solutions to the broad spectrum of text generation tasks are either already included or can be easily constructed.

- **Modularity**: Texar is designed to be highly modularized, by decoupling solutions to diverse tasks into a set of highly reusable modules. Users can construct their model at a high conceptual level just like assembling LEGO bricks. It is convenient to plug in or swap out modules, configure rich options of each module, or even switch between distinct modeling paradigms. For example, switching between maximum likelihood learning and reinforcement learning involves only minimal code changes. Modularity makes Texar useful for fast prototyping, parameter tuning, and model experimentation.

- **Extensibility**: The toolkit provides interfaces of multiple functionality levels, ranging from simple Python-like configuration files to full library APIs. Users of different needs and expertise are free to choose different interfaces for appropriate programmability and internal accessibility. The library APIs are fully compatible with the native TensorFlow interfaces, which allows a seamless integration of user-customized modules, and enables the toolkit to take advantage of the vibrant open-source community by easily importing any external components as needed.

[1] https://www.tensorflow.org

Furthermore, Texar puts much emphasis on well-structured high-quality code of uniform design patterns and consistent styles, along with clean documentations and rich tutorial examples.

In the following, we provide details of the toolkit structure and design. To demonstrate the use of the toolkit and its advantages, we perform extensive experiments and cases studies, including generalizing the state-of-the-art machine translation model to multiple text generation tasks, investigating different algorithms for language modeling, and implementing composite neural architectures beyond conventional encoder-decoder for text style transfer. All are easily realized with the versatile toolkit.

Texar is under Apache license 2.0, and will be released very soon. Please check out `http://www.cs.cmu.edu/~zhitingh` for the release progress.

2 Structure and Design

In this section, we first provide an overview of the toolkit on its design principles and overall structures. We then present the detailed structure of Texar with running examples to demonstrate the key properties of the toolkit (sec 2.2-2.4).

Figure 1 shows the stack of main modules and functionalities in Texar. Building upon the lower level deep learning platform (TensorFlow), Texar provides a comprehensive set of building blocks for model construction, training, evaluation, and prediction. Texar is designed with the goals of *versatility, modularity, and extensibility* in mind. In the following, we first present the design principles that lead to the goals (sec 2.1), and describe the detailed structure of Texar with running examples to demonstrate the properties of the toolkit (sec 2.2-2.4).

2.1 The Design of Texar

The broad variation of the many text generation tasks and the fast-growing new models and algorithms have posed unique challenges to designing a versatile toolkit. We tackle the challenges through proper decomposition of the whole experimentation pipeline, extensive sets of modules to assemble freely, and user interfaces of varying abstract levels.

Pipeline Decomposition We begin with a high-level decomposition of model construction and learning pipeline. A deep neural model is typically

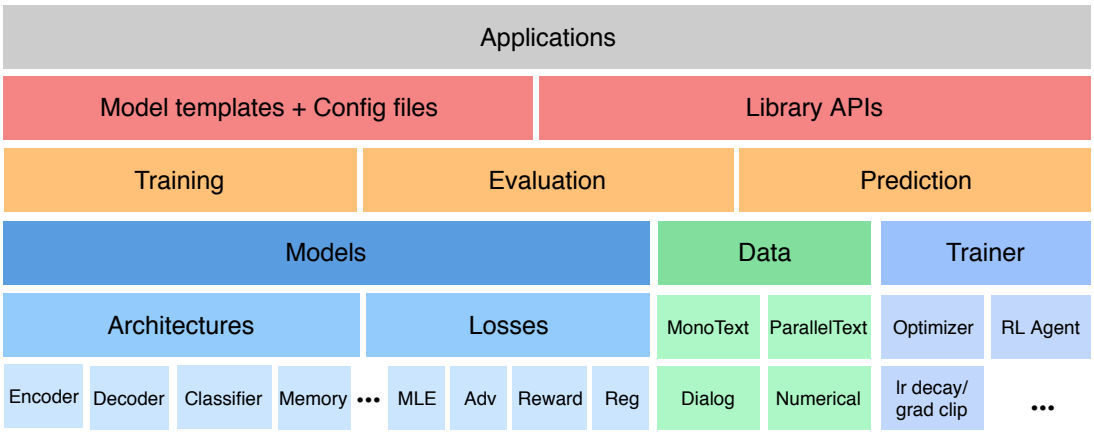

Figure 1: The stack of main modules and functionalities in Texar.

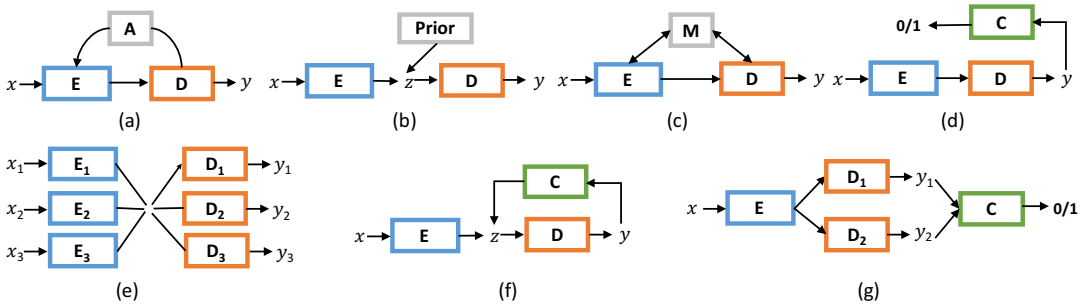

Figure 2: Example various model architectures in recent text generation literatures. `E` denotes encoder, `D` denotes decoder, `C` denotes classifier (i.e., binary discriminator). **(a)** The canonical encoder-decoder, sometimes with attentions `A` (Sutskever et al., 2014; Bahdanau et al., 2014; Luong et al., 2015b; Vaswani et al., 2017), or copy mechanisms (Gu et al., 2016; Vinyals et al., 2015a; Gulcehre et al., 2016); **(b)** Variational encoder-decoder (Bowman et al., 2015; Yang et al., 2017); **(c)** Encoder-decoder augmented with external memory (Sukhbaatar et al., 2015; Bordes et al., 2016); **(d)** Adversarial model using a binary discriminator `C`, with or without reinforcement learning (Liang et al., 2017; Zhang et al., 2017; Yu et al., 2017); **(e)** Multi-task learning with multiple encoders and/or decoders (Luong et al., 2015a; Firat et al., 2016); **(f)** Augmenting with cyclic loss (Hu et al., 2017a; Goyal et al., 2017); **(g)** Learning to align with adversary, either on samples y or hidden states (Lamb et al., 2016; Lample et al., 2017; Shen et al., 2017).

learned with the following abstract procedure:

$$\max_\theta \mathcal{L}(f_\theta, D) \tag{1}$$

where (1) f_θ is the model that defines the model architecture and the intrinsic inference procedure; (2) D is the data; (3) $\mathcal{L}$ is the losses to optimize; and (4) $\max$ denotes the optimization and learning procedure. Note that the above can have multiple losses imposed on different parts of components and parameters (e.g., generative adversarial networks (Goodfellow et al., 2014)). Texar is designed to properly decouple the four elements, and allow free combinations of them through uniform interfaces. Such design has underlay the strong modularity of the toolkit.

In particular, the decomposition of model architecture and inference (i.e., f_θ) from losses and learning has greatly improved the cleanness of the code structure and the opportunities for reuse. For example, a sequence decoder can focus solely on performing different decoding (inference) schemes, such as decoding with ground truths, and greedy, stochastic, or beam-search decoding, etc. Different learning algorithms then use different schemes as a subroutine in the learning procedure—for example, maximum likelihood learning uses decoding with ground truths (Mikolov et al., 2010), a policy gradient algorithm can use stochastic decoding (Ranzato et al., 2015), and an adversarial learning can use either the stochastic decoding for policy gradient-based updates (Yu et al., 2017) or the Gumbel-softmax reparameterized decoding (Jang et al., 2016) for direct gradient back-propagation.

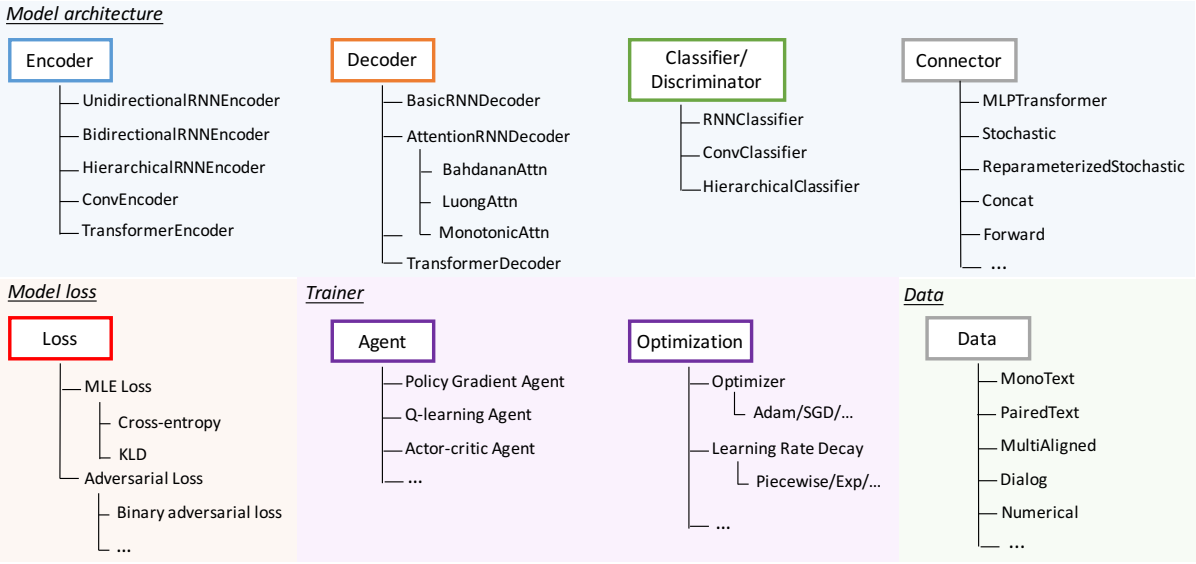

Figure 3: The catalog of a subset of modules for model construction and learning. Other modules, such as memory network modules, and those for evaluation and prediction, are omitted due to space limitations. More new modules are continuously added to the library.

With unified abstractions, the decoder and the learning algorithms need not know the implementation details of each other. This also enables convenient switch between different learning algorithms for the same model, by simply changing the inference scheme and connecting to the new learning module, without adapting the model architecture (see sec 2.3 for the example).

Modules Readily to Assemble The fast evolution of modeling and learning methodologies in the research field has led to sophisticated models that go beyond the canonical (attentive) sequence-to-sequence alike paradigms and introduce many new composite architectures. Figure 2 summarizes several model architectures recently used in the literature for different tasks. To versatilely support all these diverse approaches, we break down the complex models and extract a set of frequently-used modules (e.g., `encoders`, `decoders`, `classifiers`, etc). Figure 3 shows the catelog of a subset of modules. Crucially, Texar allows free concatenation between these modules in order to assemble arbitrary model architectures. Such concatenation can be done by directly interfacing two modules, or through an intermediate `connector` module that provides general, highly-usable functionalities of shape transformation, reparameterization (e.g., (Kingma and Welling, 2013; Jang et al., 2016)), sampling, and others.

User Interfaces It is critical for the toolkit to be flexible enough to allow construction of the simple or advanced models, while at the same time providing proper abstractions to relieve users from overly concerning about low-level implementations. To this end, Texar provides two types of user interfaces with different abstract levels: 1) Python-style configuration files that instantiate pre-defined model templates, and 2) a set of intuitive library APIs called in Python code. The former is simple, clean, straightforwardly understandable for non-expert users, and is widely adopted by other toolkits (Britz et al., 2017; Neubig et al., 2018; Klein et al., 2017), while the latter allows maximal flexibility, full access to internal states, and essentially unlimited customizability. Examples are provided in the following section.

2.2 Assemble Arbitrary Model Architectures

Figure 4 shows an example of specifying an attentive sequence-to-sequence model through either the YAML configuration file (left panel), or simple Python code (right panel), respectively.

- The configuration file passes hyperparameters to the model template which instantiates the model for subsequent training and evaluation (which are also configured through YAML). Text highlighted in blue in the figure specifies the names of modules to use. Module hyperparameters follow the module

```yaml
source_embedder: WordEmbedder
  dim: 300
encoder: UnidirectionalRNNEncoder
  rnn_cell:
    type: BasicLSTMCell
      num_units: 300
    num_layers: 1
    dropout:
      output_dropout: 0.5
      variational_recurrent: True
target_embedder: WordEmbedder
  dim: 300
decoder: AttentionRNNDecoder
  rnn_cell:
    type: BasicLSTMCell
      num_units: 300
    num_layers: 1
  attention:
    type: LuongAttention
connector: ZeroConnector
```

```python
# Read data
dataset = PairedTextData(data_hparams)
data = DataIterator(dataset).get_next()

# Encode
embedder = WordEmbedder(dataset.vocab_size, emb_dim)
encoder = UnidirectionalRNNEncoder(hparams=cell_hparams)
enc_outputs, _ = encoder(
  embedder(data['source_text_ids']), data['source_length'])

# Decode
decoder = AttentionRNNDecoder(
  memory=enc_outputs, attn_type='LuongAttention', hparams=cell_hparams)
outputs, length, _ = decoder(
  embedder(data['target_text_ids']), data['target_length']-1, mode='greedy_train')

# Loss
loss = sequence_sparse_softmax_cross_entropy(
  labels=data['target_text_ids'][:,1:], logits=outputs.logits, seq_length=length)
```

Figure 4: Two ways of specifying an attentive sequence-to-sequence model. **Left:** Snippet of an example YAML configuration file of the sequence-to-sequence model template. Only those hyperparameters that the user concerns are specified explicitly in the particular file, while the remaining many hyperparameters can be omitted and will take default values. **Right:** Python code assembling the sequence-to-sequence model, using the Texar library APIs. Modules are created as Python objects, and then can be called as functions to perform the main logic (e.g., decoding) of the module. (Other code such as optimization is omitted.)

names as children in the configuration hierarchy. Note that most of the hyperparameters have sensible default values, and users only have to specify a small subset of them. Hyperparameters taking default values can be omitted in the configuration file.

- The library APIs offer high-level function calls. Users are enabled to efficiently build desired pipelines at a high conceptual level, without worrying too much about the low-level implementations. Power users are also given the option to access the full internal states for native programming and low-level manipulations.

2.3 Plug-in and Swap-out Modules

Texar builds a shared abstraction of the broad set of text generation tasks, and creates highly reusable modules. It is thus very convenient to switch between different application contexts, or change from one modeling paradigm to another, by simply plugging in/swapping out a single or few modules, or even merely changing a configuration parameter, while keeping other parts of the modeling and training pipeline agnostic.

Figure 5 illustrates an example of switching between three major learning paradigms of an RNN decoder, i.e., maximum-likelihood based supervised learning, adversarial learning, and reinforcement learning, using the library APIs. Local modification of only few lines of code is enough to achieve such change. In particular, the same decoder is called with different decoding modes (e.g., `greed_train` and `greedy_infer`), and discriminator or reinforcement learning agent is added when needed, with simple API calls.

The convenient module replacement can be valuable for fast exploration of different algorithms for a specific task, or quick experimentation of an algorithm's generalization on different tasks.

2.4 Customize with Extensible Interfaces

With the aim of supporting the rapidly advancing research area of text generation, Texar emphasizes heavily on extensibility, and allows easy addition of customized or external modules through various interfaces, without editing the Texar codebase.

With the YAML configuration file, users can directly insert their own modules by providing the Python importing path to the module. For example, to use a externally implemented RNN cell in

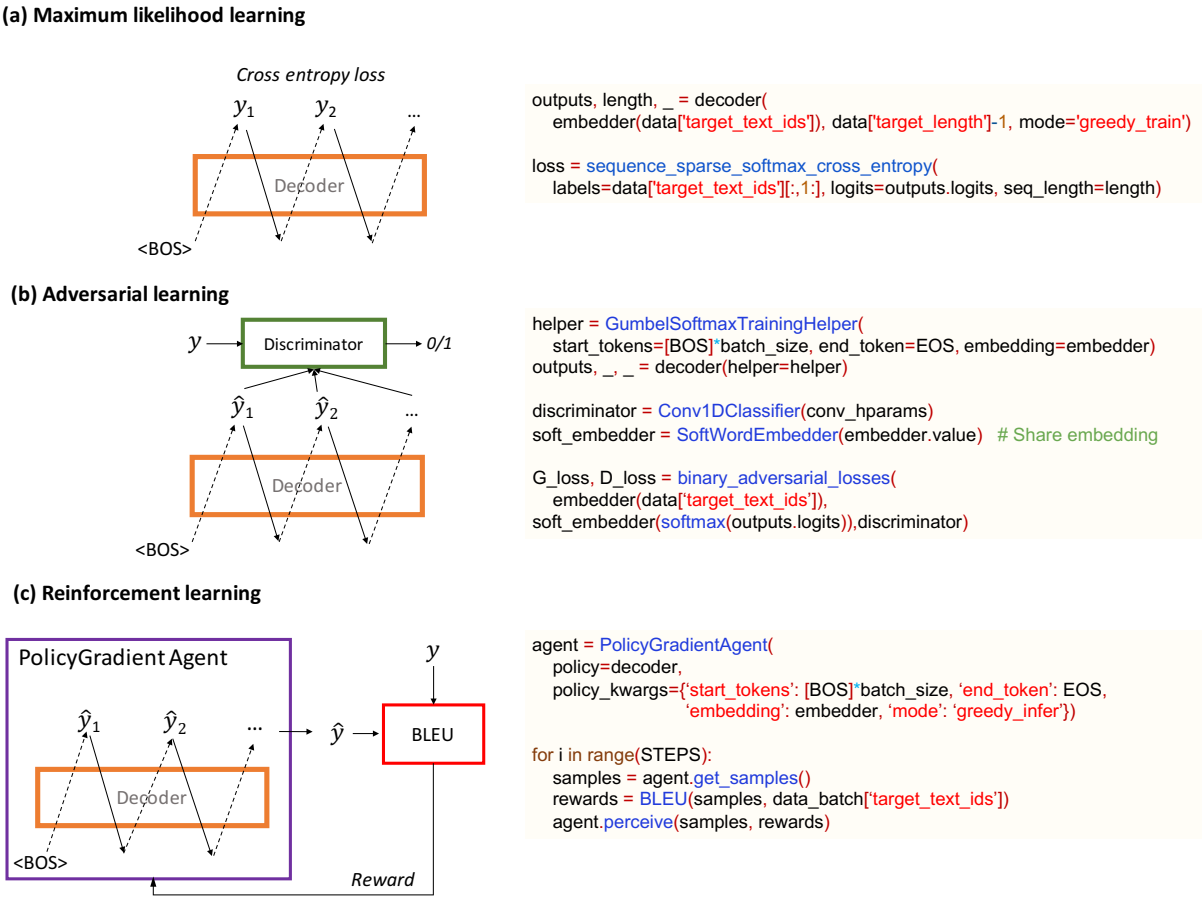

Figure 5: Switching between the different learning paradigms of a decoder involves only modification of Line.14-19 in the right panel of Figure 4. In particular, the same decoder is called with different decoding modes (schemes), and the discriminator or reinforcement learning agent is added when needed, with simple API calls. **Left:** The module structure of each of the paradigms; **Right:** The respective code. For the adversarial learning in (b), the continuous Gumbel-softmax approximation (Jang et al., 2016) (with `GumbelSoftmaxTrainingHelper`) to the generated sample is used to enable gradient propagation from the discriminator to the decoder.

the sequence-to-sequence model encoder, one can simply change Lines.5-6 in the left panel of Figure 4 to the following:

```
5  type: path.to.MyCell
6      num_units: 300
7      some_new_arg: 123
8      ...
```

as long as the `MyCell` class is accessible by Python, and its interface is compatible to other parts of the model.

Incorporating customized modules with Texar library APIs is even more flexible and straightforward. As the library APIs are designed to be coherent with the native TensorFlow programming interfaces, any externally-defined modules can be seamlessly combined with Texar components to build arbitrary complex models and pipelines.

3 Experiments

We perform extensive experiments to demonstrate the use and advantage of Texar. In particular, we conduct case studies on *technique sharing* that is uniquely supported by our toolkit: (1) We deploy the state-of-the-art machine translation model on other tasks to study its generality, and obtain improved performance over previous methods; (2) We apply various model paradigms on the task of language modeling to compare the different methods. Besides, to further demonstrate the versatility of Texar, we show a case study on the newly-emerging task of text style transfer, which typically involves composite neural architectures beyond the conventional encoder-decoder.

Task: VAE language modeling			
Dataset	**Metrics**	**VAE-LSTM**	**VAE-Transformer**
Yahoo (Yang et al., 2017)	Test PPL	68.31	**61.26**
	Test NLL	337.36	**328.67**
PTB (Bowman et al., 2015)	Test PPL	105.27	**102.46**
	Test NLL	102.06	**101.46**

Table 1: Comparison of Transformer decoder and LSTM RNN decoder on VAE language modeling (Bowman et al., 2015). Test set perplexity (PPL) and sentence-level negative log likelihood (NLL) are evaluated (The lower the better).

3.1 One Technique on Many Tasks: Transformer

Transformer (Vaswani et al., 2017) is a recently developed model that achieves state-of-the-art performance on machine translation. Different from the widely-used attentive sequence-to-sequence models (Bahdanau et al., 2014), Transformer introduces a new *self-attention* technique in which each generated token attends to all previously generated tokens. It would be interesting to see how the technique generalizes to other text generation tasks beyond machine translation. We deploy the self-attention Transformer decoder on two tasks, namely, variational autoencoder (VAE) based language modeling (Bowman et al., 2015) and conversation generation (Serban et al., 2016).

The first task is to use the VAE model (Kingma and Welling, 2013) for language modeling. LSTM RNN has been widely-used in VAE for decoding sentences. We follow the experimental setting in previous work (Bowman et al., 2015; Yang et al., 2017), and test two models, one with the LSTM RNN decoder, and the other with the Transformer decoder. All other configurations (including the encoders) are the same in the two models. Changing the decoder in the whole experiment pipeline is easily achieved on Texar, thanks to the modularized design. Both the LSTM decoder and the Transformer decoder have around 6.3M free parameters to learn. Table 1 shows the results. We see that the VAE with Transformer decoder consistently improves over the VAE with conventional LSTM decoder.

The second task is to generate response given a conversation history. We use the popular hierarchical recurrent encoder-decoder model (HRED) (Serban et al., 2016) as the base model, which treats a conversation as a transduction task. The conversation history is seen as the source sequence and is modeled with a hierarchical encoder. Each utterance in the dialog history is first encoded with a first-level RNN. The resulting hidden states of the sequence of utterance are then encoded with a second-level RNN. We follow the experimental setting in (Zhao et al., 2017). In particular, the first-level RNN is set to be bidirectional and the second-level is unidirectional. Such configuration is easily implemented by setting the hyperparameters of the Texar `HierarchicalRNNEncoder`. Similar to the above task, we compare two models, one with an GRU RNN decoder as in the original work, and the other with an Transformer decoder. Table 2 shows the results. Again, we see that the Transformer model generalizes well to the conversation generation setting, and consistently outperforms the GRU RNN counterpart.

Task: Conversation generation		
Metrics	**HERD-GRU**	**HERD-Tnsfmr**
BLEU-3 prec	0.281	**0.289**
BLEU-3 recall	0.256	**0.273**
BLEU-4 prec	0.228	**0.232**
BLEU-4 recall	0.205	**0.214**

Table 2: Comparison of Transformer decoder and GRU RNN decoder on conversation generation within the HERD model (Bowman et al., 2015). The Switchboard data (Zhao et al., 2017) is used.

3.2 One Task with Many Techniques: Language Modeling

We next showcase how Texar can support investigation of diverse techniques on a single task. This can be valuable for research community to standardize experimental configurations and foster fair, reproducible comparisons. As as case study, we choose the standard language modeling task (Zaremba et al., 2014). Note that this is differ-

Models	Test PPL
LSTM RNN with MLE (Zaremba et al., 2014)	74.23
LSTM RNN with seqGAN (Yu et al., 2017)	74.12
Memory Network LM (Sukhbaatar et al., 2015)	94.82

Table 3: Comparison of the three models on the task of language modeling, using the PTB dataset (Zaremba et al., 2014).

Models	Accuracy	BLEU
Shen et al. (2017)	79.5	12.4
Shen et al. (2017) on Texar	82.5	13.0
Hu et al. (2017a) on Texar	**88.6**	**38.0**

Table 4: Text style transfer on the Yelp data (Shen et al., 2017). The first row is the original open-source implementation by the author (Shen et al., 2017). The subsequent two rows are Texar implementations of the two work.

ent from the VAE language modeling task above, due to different data partition strategies conventionally adopted in respective research lines.

We compare three models as shown in Table 3. The LSTM RNN trained with the maximum likelihood estimation (MLE) (Zaremba et al., 2014) is the most widely used model for language modeling, due to its simplicity and prominent performance. We use the exact same architecture as generator and setup a (seq)GAN (Yu et al., 2017) system to train the language model with adversarial learning. (The generator is pre-trained with MLE.) From Table 3 we see that adversarial learning does not improve the perplexity. This is partly because of the high variance of the policy gradient in seqGAN learning. Besides, test set perplexity is not a perfect metric for evaluating language modeling, though it is the most widely-used metrics in the field. We further evaluate a memory network-based language model (Sukhbaatar et al., 2015) which has the same number of free parameters (11M) with the LSTM RNN model. The test set perplexity is significantly higher than the LSTM RNNs, which is not unreasonable because LSTM RNN models are well studied for language modeling and a number of optimal modeling and optimization choices are already known.

3.3 Text Style Transfer

To further demonstrate the versatility of Texar for composing complicated model architectures, we next choose the the newly emerging task of text style transfer (Hu et al., 2017a; Shen et al., 2017). The task aims to manipulate the text of an input sentence to change from one style to another (e.g., from positive sentiment to negative), given only non-parallel training data of each style. The criteria is that the resulting sentence accurately entails the target style, while preserving the content and other properties well.

We use Texar to implement the models from both (Hu et al., 2017a) and (Shen et al., 2017), whose model architectures fall in the category (f) and (g) in Figure 2, respectively. Experimental settings mostly follow those in (Shen et al., 2017). Following previous setting, we use a pre-trained sentiment classifier to evaluate the transferred style accuracy. For evaluating how well the generated sentence preserves the original content, we measure the BLEU score between the generated sentence and the respective original one (The higher the better). Note that we do not mean to perform exhaustive evaluations of the methods, but instead aim to demonstrate the flexibility of the toolkit for implementing different composite model architectures beyond conventional encoder-decoder. Table 4 shows the results. Our re-implementation of (Shen et al., 2017) recovers and slightly surpasses the original results, while the implementation of (Hu et al., 2017a) provides the best performance in terms of the two metrics.

4 Related Work

Text generation is a broad research area with rapid advancement. Figure 2 summarizes some popular and emerging models used in the diverse contexts of the field. There are some existing toolkits that focus on tasks of neural machine translation and alike, such as Google Seq2seq (Britz et al., 2017) and Tensor2Tensor (Vaswani et al., 2018) on TensorFlow, OpenNMT (Klein et al., 2017) on (Py)Torch, XNMT (Neubig et al., 2018) on DyNet, and Nematus (Sennrich et al., 2017) on Theano, and MarianNMT (Junczys-Dowmunt et al., 2018) on C++. ParlAI (Miller et al., 2017) is a software platform specialized for dialog research. Differing from these task-specific toolkits, Texar aims to cover as many text generation tasks as possible. The goal of versatility poses unique challenges to the design. We combat the challenges through proper pipeline decomposition, ready-to-assemble modules, and user interfaces of varying abstract levels.

There are also libraries for general NLP appli-

cations (AllenAI; Pytorch; DMLC). With the focus on text generation, we provide a more comprehensive and readily-usable modules and functionalities to relevant tasks, enable users to efficiently build their pipelines at a high conceptual level without worrying too much on low-level details. Some platforms exist for specific types of algorithms, such as OpenAI Gym (Brockman et al., 2016), DeepMind Control Suite (Tassa et al., 2018), and ELF (Tian et al., 2017) for reinforcement learning in game environments. Texar has drawn inspirations from these toolkits when designing relevant specific algorithm supports.

5 Conclusion and Future Work

This paper has introduced Texar, a text generation toolkit that is designed to be versatile to support the broad set of applications and algorithms, to be modularized to enable easy replacement of any components, and to be extensible to allow seamless integration of any external modules. Features and functionalities will continue be added to the toolkit, including distributed model training, service deployment, more model building blocks, and more applications related to text generation or beyond. We invite researchers and practitioners to join and enrich the toolkit, and in the end help push forward the text generation research and applications together.

References

AllenAI. AllenNLP. Website: http://allennlp.org.

Dzmitry Bahdanau, Philemon Brakel, Kelvin Xu, Anirudh Goyal, Ryan Lowe, Joelle Pineau, Aaron Courville, and Yoshua Bengio. 2016. An actor-critic algorithm for sequence prediction. *arXiv preprint arXiv:1607.07086*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Antoine Bordes, Y-Lan Boureau, and Jason Weston. 2016. Learning end-to-end goal-oriented dialog. *arXiv preprint arXiv:1605.07683*.

Samuel R Bowman, Luke Vilnis, Oriol Vinyals, Andrew M Dai, Rafal Jozefowicz, and Samy Bengio. 2015. Generating sentences from a continuous space. *arXiv preprint arXiv:1511.06349*.

Denny Britz, Anna Goldie, Thang Luong, and Quoc Le. 2017. Massive exploration of neural machine translation architectures. *arXiv preprint arXiv:1703.03906*.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. Openai gym. *arXiv preprint arXiv:1606.01540*.

Peter F Brown, John Cocke, Stephen A Della Pietra, Vincent J Della Pietra, Fredrick Jelinek, John D Lafferty, Robert L Mercer, and Paul S Roossin. 1990. A statistical approach to machine translation. *Computational linguistics*, 16(2):79–85.

DMLC. GluonNLP.

Orhan Firat, Kyunghyun Cho, and Yoshua Bengio. 2016. Multi-way, multilingual neural machine translation with a shared attention mechanism. *arXiv preprint arXiv:1601.01073*.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

Anirudh Goyal Alias Parth Goyal, Alessandro Sordoni, Marc-Alexandre Côté, Nan Ke, and Yoshua Bengio. 2017. Z-forcing: Training stochastic recurrent networks. In *Advances in Neural Information Processing Systems*, pages 6716–6726.

Jiatao Gu, Zhengdong Lu, Hang Li, and Victor OK Li. 2016. Incorporating copying mechanism in sequence-to-sequence learning. *arXiv preprint arXiv:1603.06393*.

Caglar Gulcehre, Sungjin Ahn, Ramesh Nallapati, Bowen Zhou, and Yoshua Bengio. 2016. Pointing the unknown words. *arXiv preprint arXiv:1603.08148*.

Eduard Hovy and Chin-Yew Lin. 1998. Automated text summarization and the summarist system. In *Proceedings of a workshop on held at Baltimore, Maryland: October 13-15, 1998*, pages 197–214. Association for Computational Linguistics.

Zhiting Hu, Zichao Yang, Xiaodan Liang, Ruslan Salakhutdinov, and Eric P Xing. 2017a. Toward controlled generation of text. In *International Conference on Machine Learning*.

Zhiting Hu, Zichao Yang, Ruslan Salakhutdinov, and Eric P Xing. 2017b. On unifying deep generative models. *arXiv preprint arXiv:1706.00550*.

Eric Jang, Shixiang Gu, and Ben Poole. 2016. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*.

Marcin Junczys-Dowmunt, Roman Grundkiewicz, Tomasz Dwojak, Hieu Hoang, Kenneth Heafield, Tom Neckermann, Frank Seide, Ulrich Germann, Alham Fikri Aji, Nikolay Bogoychev, Andr F. T. Martins, and Alexandra Birch. 2018. Marian: Fast neural machine translation in c++. *arXiv preprint arXiv:1804.00344*.

Andrej Karpathy and Li Fei-Fei. 2015. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3128–3137.

Diederik P Kingma and Max Welling. 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander M Rush. 2017. Opennmt: Open-source toolkit for neural machine translation. *arXiv preprint arXiv:1701.02810*.

Alex M Lamb, Anirudh Goyal Alias Parth Goyal, Ying Zhang, Saizheng Zhang, Aaron C Courville, and Yoshua Bengio. 2016. Professor forcing: A new algorithm for training recurrent networks. In *Advances In Neural Information Processing Systems*, pages 4601–4609.

Guillaume Lample, Ludovic Denoyer, and Marc'Aurelio Ranzato. 2017. Unsupervised machine translation using monolingual corpora only. *arXiv preprint arXiv:1711.00043*.

Xiaodan Liang, Zhiting Hu, Hao Zhang, Chuang Gan, and Eric P Xing. 2017. Recurrent topic-transition gan for visual paragraph generation. *CoRR, abs/1703.07022*, 2.

Minh-Thang Luong, Quoc V Le, Ilya Sutskever, Oriol Vinyals, and Lukasz Kaiser. 2015a. Multi-task sequence to sequence learning. *arXiv preprint arXiv:1511.06114*.

Minh-Thang Luong, Hieu Pham, and Christopher D Manning. 2015b. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.

Nitin Madnani and Bonnie J Dorr. 2010. Generating phrasal and sentential paraphrases: A survey of data-driven methods. *Computational Linguistics*, 36(3):341–387.

Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Eleventh Annual Conference of the International Speech Communication Association*.

Alexander H Miller, Will Feng, Adam Fisch, Jiasen Lu, Dhruv Batra, Antoine Bordes, Devi Parikh, and Jason Weston. 2017. Parlai: A dialog research software platform. *arXiv preprint arXiv:1705.06476*.

Graham Neubig, Matthias Sperber, Xinyi Wang, Matthieu Felix, Austin Matthews, Sarguna Padmanabhan, Ye Qi, Devendra Singh Sachan, Philip Arthur, Pierre Godard, et al. 2018. Xnmt: The extensible neural machine translation toolkit. *arXiv preprint arXiv:1803.00188*.

Pytorch. QuickNLP.

Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. 2015. Sequence level training with recurrent neural networks. *arXiv preprint arXiv:1511.06732*.

Abigail See, Peter J Liu, and Christopher D Manning. 2017. Get to the point: Summarization with pointer-generator networks. *arXiv preprint arXiv:1704.04368*.

Rico Sennrich, Orhan Firat, Kyunghyun Cho, Alexandra Birch, Barry Haddow, Julian Hitschler, Marcin Junczys-Dowmunt, Samuel Läubli, Antonio Valerio Miceli Barone, Jozef Mokry, et al. 2017. Nematus: a toolkit for neural machine translation. *arXiv preprint arXiv:1703.04357*.

Iulian Vlad Serban, Alessandro Sordoni, Yoshua Bengio, Aaron C Courville, and Joelle Pineau. 2016. Building end-to-end dialogue systems using generative hierarchical neural network models.

Tianxiao Shen, Tao Lei, Regina Barzilay, and Tommi Jaakkola. 2017. Style transfer from non-parallel text by cross-alignment. In *Advances in Neural Information Processing Systems*, pages 6833–6844.

Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. 2015. End-to-end memory networks. In *Advances in neural information processing systems*, pages 2440–2448.

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.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. 2018. Deepmind control suite. *arXiv preprint arXiv:1801.00690*.

Yuandong Tian, Qucheng Gong, Wenling Shang, Yuxin Wu, and C Lawrence Zitnick. 2017. Elf: An extensive, lightweight and flexible research platform for real-time strategy games. In *Advances in Neural Information Processing Systems*, pages 2656–2666.

Ashish Vaswani, Samy Bengio, Eugene Brevdo, Francois Chollet, Aidan N Gomez, Stephan Gouws, Llion Jones, Łukasz Kaiser, Nal Kalchbrenner, Niki Parmar, et al. 2018. Tensor2tensor for neural machine translation. *arXiv preprint arXiv:1803.07416*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 6000–6010.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015a. Pointer networks. In *Advances in Neural Information Processing Systems*, pages 2692–2700.

Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. 2015b. Show and tell: A neural image caption generator. In *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*, pages 3156–3164. IEEE.

Jason D Williams and Steve Young. 2007. Partially observable markov decision processes for spoken dialog systems. *Computer Speech & Language*, 21(2):393–422.

Sam Wiseman, Stuart M Shieber, and Alexander M Rush. 2017. Challenges in data-to-document generation. *arXiv preprint arXiv:1707.08052*.

Zichao Yang, Zhiting Hu, Ruslan Salakhutdinov, and Taylor Berg-Kirkpatrick. 2017. Improved variational autoencoders for text modeling using dilated convolutions. *arXiv preprint arXiv:1702.08139*.

Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. 2017. Seqgan: Sequence generative adversarial nets with policy gradient. In *AAAI*, pages 2852–2858.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. 2014. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*.

Yizhe Zhang, Zhe Gan, Kai Fan, Zhi Chen, Ricardo Henao, Dinghan Shen, and Lawrence Carin. 2017. Adversarial feature matching for text generation. *arXiv preprint arXiv:1706.03850*.

Tiancheng Zhao, Ran Zhao, and Maxine Eskenazi. 2017. Learning discourse-level diversity for neural dialog models using conditional variational autoencoders. *arXiv preprint arXiv:1703.10960*.