TextBox

Release 0.2.1

AI Box

Jun 23, 2022

API REFERENCE:

HomePage | Paper | Docs | Models | Datasets

CHAPTER

ONE

TEXTBOX.CONFIG

1.1 textbox.config.configurator

class textbox.config.configurator.**Config**(*model=None*, *dataset=None*, *config_file_list=None*, *config_dict=None*)

Bases: `object`

Configurator module that load the defined parameters.

Configurator module will first load the default parameters from the fixed properties in TextBox and then load parameters from the external input.

External input supports three kind of forms: config file, command line and parameter dictionaries.

- config file: It's a file that record the parameters to be modified or added. It should be in `yaml` format, e.g. a config file is 'example.yaml', the content is:

  learning_rate: 0.001

  train_batch_size: 2048

- command line: It should be in the format as '--learning_rate=0.001'
- parameter dictionaries: It should be a dict, where the key is parameter name and the value is parameter value, e.g. config_dict = {'learning_rate': 0.001}

Configuration module allows the above three kind of external input format to be used together, the priority order is as following:

command line > parameter dictionaries > config file (model > dataset > overall)

e.g. If we set learning_rate=0.01 in config file, learning_rate=0.02 in command line, learning_rate=0.03 in parameter dictionaries.

Finally the learning_rate is equal to 0.02.

CHAPTER

TWO

TEXTBOX.DATA

2.1 textbox.data.dataloader

2.1.1 textbox.data.dataloader.abstract_dataloader

class textbox.data.dataloader.abstract_dataloader.**AbstractDataLoader**(*config*, *dataset*, *batch_size=1*, *shuffle=False*, *drop_last=True*, *DDP=False*)

Bases: `object`

***AbstractDataLoader* is an abstract object which would return a batch of data.**

And it is also the ancestor of all other dataloader.

Parameters

- **config** (*Config*) – The config of dataloader.
- **dataset** (*Corpus*) – The corpus for partition of dataset.
- **batch_size** (*int, optional*) – The batch_size of dataloader. Defaults to 1.
- **shuffle** (*bool*) – If `True`, dataloader will shuffle before every epoch.

dataset

The necessary elements of this dataloader.

Type
dict

pr

Pointer of dataloader.

Type
int

step

The increment of *pr* for each batch.

Type
int

batch_size

The max interaction number for all batch.

Type
int

get_reference()

Get reference documents for current data loader return is supposed to be reference_corpus as list -> list -> word

2.1.2 textbox.data.dataloader.single_sent_dataloader

class textbox.data.dataloader.single_sent_dataloader.**SingleSentenceDataLoader**(*config*, *dataset*, *batch_size=1*, *shuffle=False*, *drop_last=True*, *DDP=False*)

Bases: *AbstractDataLoader*

GeneralDataLoader is used for general model and it just return the origin data.

Parameters

- **config** (Config) – The config of dataloader.
- **dataset** (SingleSentenceDataset) – The dataset of dataloader. Corpus, see textbox.data.corpus for more details
- **batch_size** (*int, optional*) – The batch_size of dataloader. Defaults to 1.
- **shuffle** (*bool, optional*) – Whether the dataloader will be shuffle after a round. Defaults to False.

2.1.3 textbox.data.dataloader.paired_sent_dataloader

class textbox.data.dataloader.paired_sent_dataloader.**CopyPairedSentenceDataLoader**(*config*, *dataset*, *batch_size=1*, *shuffle=False*, *drop_last=True*, *DDP=False*)

Bases: *PairedSentenceDataLoader*

static get_extra_zeros(*oovs*)

class textbox.data.dataloader.paired_sent_dataloader.**PairedSentenceDataLoader**(*config*, *dataset*, *batch_size=1*, *shuffle=False*, *drop_last=True*, *DDP=False*)

Bases: *AbstractDataLoader*

GeneralDataLoader is used for general model and it just return the origin data.

Parameters

- **config** (Config) – The config of dataloader.
- **dataset** (PairedSentenceDataset) – The dataset of dataloader. Corpus, see textbox.data.corpus for more details
- **batch_size** (*int, optional*) – The batch_size of dataloader. Defaults to 1.
- **shuffle** (*bool, optional*) – Whether the dataloader will be shuffle after a round. Defaults to False.

2.1.4 textbox.data.dataloader.attr_sent_dataloader

class textbox.data.dataloader.attr_sent_dataloader.**AttributedSentenceDataLoader**(*config*, *dataset*, *batch_size=1*, *shuffle=False*, *drop_last=True*, *DDP=False*)

Bases: *AbstractDataLoader*

GeneralDataLoader is used for general model and it just return the origin data.

Parameters

- **config** (Config) – The config of dataloader.
- **dataset** (AttributedSentenceDataset) – The dataset of dataloader. Corpus, see textbox.data.corpus for more details
- **batch_size** (*int, optional*) – The batch_size of dataloader. Defaults to 1.
- **shuffle** (*bool, optional*) – Whether the dataloader will be shuffle after a round. Defaults to False.

2.2 textbox.data.dataset

2.2.1 textbox.data.dataset.abstract_dataset

class textbox.data.dataset.abstract_dataset.**AbstractDataset**(*config*)

Bases: object

***AbstractDataset* is an abstract object which stores the original dataset in memory.**
And it is also the ancestor of all other dataset.

Parameters
config (Config) – Global configuration object.

2.2.2 textbox.data.dataset.single_sent_dataset

class textbox.data.dataset.single_sent_dataset.**SingleSentenceDataset**(*config*)

Bases: *AbstractDataset*

2.2.3 textbox.data.dataset.paired_sent_dataset

class textbox.data.dataset.paired_sent_dataset.**CopyPairedSentenceDataset**(*config*)

Bases: *PairedSentenceDataset*

static text2idx(*source_text*, *target_text*, *token2idx*, *sos_idx*, *eos_idx*, *unk_idx*, *is_pgen=False*)

class textbox.data.dataset.paired_sent_dataset.**PairedSentenceDataset**(*config*)

Bases: *AbstractDataset*

2.2.4 textbox.data.dataset.attr_sent_dataset

class textbox.data.dataset.attr_sent_dataset.**AttributedSentenceDataset**(*config*)

Bases: *AbstractDataset*

2.3 textbox.data.utils

textbox.data.utils.**attribute2idx**(*text*, *token2idx*)

transform attribute to id.

Parameters

- **text** (*List[List[List[str]]] or List[List[List[List[str]]]]*) – list of attribute data, consisting of multiple groups.
- **token2idx** (*dict*) – map token to index

Returns

attribute index length (None or List[List[int]]): sequence length

Return type

idx (List[List[List[int]]] or List[List[List[List[int]]]])

textbox.data.utils.**build_attribute_vocab**(*text*)

Build attribute vocabulary of list of attribute data.

Parameters

text (*List[List[List[str]]] or List[List[List[List[str]]]]*) – list of attribute data, consisting of multiple groups.

Returns

- idx2token (dict): map index to token.
- token2idx (dict): map token to index.

Return type

tuple

textbox.data.utils.**build_vocab**(*text*, *max_vocab_size*, *special_token_list*)

Build vocabulary of list of text data.

Parameters

- **text** (*List[List[List[str]]] or List[List[List[List[str]]]]*) – list of text data, consisting of multiple groups.
- **max_vocab_size** (*int*) – max size of vocabulary.
- **special_token_list** (*List[str]*) – list of special tokens.

Returns

- idx2token (dict): map index to token.
- token2idx (dict): map token to index.
- max_vocab_size (int): updated max size of vocabulary.

Return type
tuple

textbox.data.utils.**construct_quick_test_dataset**(*dataset_path*)

textbox.data.utils.**data_preparation**(*config*, *save=False*)

call *dataloader_construct()* to create corresponding dataloader.

Parameters

- **config** (Config) – An instance object of Config, used to record parameter information.
- **save** (*bool, optional*) – If `True`, it will call `save_datasets()` to save split dataset. Defaults to `False`.

Returns

- train_data (AbstractDataLoader): The dataloader for training.
- valid_data (AbstractDataLoader): The dataloader for validation.
- test_data (AbstractDataLoader): The dataloader for testing.

Return type
tuple

textbox.data.utils.**dataloader_construct**(*name*, *config*, *dataset*, *batch_size=1*, *shuffle=False*, *drop_last=True*, *DDP=False*)

Get a correct dataloader class by calling *get_dataloader()* to construct dataloader.

Parameters

- **name** (*str*) – The stage of dataloader. It can only take two values: 'train' or 'evaluation'.
- **config** (Config) – An instance object of Config, used to record parameter information.
- **dataset** (*Dataset or list of Dataset*) – The split dataset for constructing dataloader.
- **batch_size** (*int, optional*) – The batch_size of dataloader. Defaults to 1.
- **shuffle** (*bool, optional*) – Whether the dataloader will be shuffle after a round. Defaults to `False`.
- **drop_last** (*bool, optional*) – Whether the dataloader will drop the last batch. Defaults to `True`.

- **DDP** (*bool, optional*) – Whether the dataloader will distribute in different GPU. Defaults to `False`.

Returns
Constructed dataloader in split dataset.

Return type
AbstractDataLoader or list of AbstractDataLoader

`textbox.data.utils.`**deconstruct_quick_test_dataset**(*dataset_path*)

`textbox.data.utils.`**get_dataloader**(*config*)

Return a dataloader class according to `config` and `split_strategy`.

Parameters
config (`Config`) – An instance object of Config, used to record parameter information.

Returns
The dataloader class that meets the requirements in `config` and `split_strategy`.

Return type
type

`textbox.data.utils.`**get_dataset**(*config*)

Create dataset according to `config['model']` and `config['MODEL_TYPE']`.

Parameters
config (`Config`) – An instance object of Config, used to record parameter information.

Returns
Constructed dataset.

Return type
Dataset

`textbox.data.utils.`**load_data**(*dataset_path*, *tokenize_strategy*, *max_length*, *language*, *multi_sentence*, *max_num*)

Load dataset from split (train, valid, test). This is designed for single sentence format.

Parameters

- **dataset_path** (*str*) – path of dataset dir.
- **tokenize_strategy** (*str*) – strategy of tokenizer.
- **max_length** (*int*) – max length of sequence.
- **language** (*str*) – language of text.
- **multi_sentence** (*bool*) – whether to split text into sentence level.
- **max_num** (*int*) – max number of sequence.

Returns
the text list loaded from dataset path.

Return type
List[List[str]]

`textbox.data.utils.`**pad_sequence**(*idx*, *length*, *padding_idx*, *num=None*)

padding a batch of word index data, to make them have equivalent length

Parameters

- **idx** (*List[List[int]] or List[List[List[int]]]*) – word index

- **length** (*List[int] or List[List[int]]*) – sequence length
- **padding_idx** (*int*) – the index of padding token
- **num** (*List[int]*) – sequence number

Returns
word index length (List[int] or List[List[int]]): sequence length num (List[int]): sequence number

Return type
idx (List[List[int]] or List[List[List[int]]])

textbox.data.utils.**text2idx**(*text*, *token2idx*, *tokenize_strategy*)

transform text to id and add sos and eos token index.

Parameters

- **text** (*List[List[List[str]]] or List[List[List[List[str]]]]*) – list of text data, consisting of multiple groups.
- **token2idx** (*dict*) – map token to index
- **tokenize_strategy** (*str*) – strategy of tokenizer.

Returns
word index length (List[List[int]] or List[List[List[int]]]): sequence length num (None or List[List[int]]): sequence number

Return type
idx (List[List[List[int]]] or List[List[List[List[int]]]])

textbox.data.utils.**tokenize**(*text*, *tokenize_strategy*, *language*, *multi_sentence*)

Tokenize text data.

Parameters

- **text** (*str*) – text data.
- **tokenize_strategy** (*str*) – strategy of tokenizer.
- **language** (*str*) – language of text.
- **multi_sentence** (*bool*) – whether to split text into sentence level.

Returns
the tokenized text data.

Return type
List[str]

CHAPTER

THREE

TEXTBOX.EVALUATOR

3.1 textbox.evaluator.averagelength_evaluator

class textbox.evaluator.averagelength_evaluator.**AvgLenEvaluator**

Bases: AbstractEvaluator

3.2 textbox.evaluator.bertscore_evaluator

class textbox.evaluator.bertscore_evaluator.**BertScoreEvaluator**(*model*, *num_layers*)

Bases: AbstractEvaluator

Bert Score Evaluator. Now, we support metrics *'bert score'*.

3.3 textbox.evaluator.bleu_evaluator

class textbox.evaluator.bleu_evaluator.**BleuEvaluator**(*task_type*)

Bases: AbstractEvaluator

Bleu Evaluator. Now, we support metrics *'bleu'*

3.4 textbox.evaluator.chrf++_evaluator

class textbox.evaluator.chrfplusplus_evaluator.**ChrfPlusPlusEvaluator**

Bases: AbstractEvaluator

3.5 textbox.evaluator.cider_evaluator

class textbox.evaluator.cider_evaluator.**CIDErEvaluator**

Bases: AbstractEvaluator

CIDEr Evaluator. Now, we support metrics *'CIDEr'*.

3.6 textbox.evaluator.distinct_evaluator

class textbox.evaluator.distinct_evaluator.**DistinctEvaluator**

Bases: AbstractEvaluator

Distinct Evaluator. Now, we support metrics *'inter-distinct'* and *'intra-distinct'*.

dist_func(*generate_sentence*, *ngram*)

3.7 textbox.evaluator.meteor_evaluator

class textbox.evaluator.meteor_evaluator.**MeteorEvaluator**

Bases: AbstractEvaluator

3.8 textbox.evaluator.selfbleu_evaluator

class textbox.evaluator.selfbleu_evaluator.**SelfBleuEvaluator**

Bases: AbstractEvaluator

Bleu Evaluator. Now, we support metrics *'self-bleu'*.

3.9 textbox.evaluator.unique_evaluator

class textbox.evaluator.unique_evaluator.**UniqueEvaluator**

Bases: AbstractEvaluator

Unique Evaluator. Now, we support metrics *'unique'*.

CHAPTER

FOUR

TEXTBOX.MODEL

4.1 textbox.model.abstract_generator

class textbox.model.abstract_generator.**AbstractModel**(*config*, *dataset*)

Bases: Module

Base class for all models

generate(*batch_data*, *eval_data*)

Predict the texts conditioned on a noise or sequence.

Parameters

- **batch_data** (*Corpus*) – Corpus class of a single batch.
- **eval_data** – Common data of all the batches.

Returns
Generated text, shape: [batch_size, max_len]

Return type
torch.Tensor

training: bool

class textbox.model.abstract_generator.**AttributeGenerator**(*config*, *dataset*)

Bases: *AbstractModel*

This is a abstract general attribute generator. All the attribute model should implement this class. The base general attribute generator class provide the basic parameters information.

training: bool

type = 4

class textbox.model.abstract_generator.**GenerativeAdversarialNet**(*config*, *dataset*)

Bases: *UnconditionalGenerator*

This is a abstract general generative adversarial network. All the GAN model should implement this class. The base general generative adversarial network class provide the basic parameters information.

calculate_d_train_loss(*real_data*, *fake_data*)

Calculate the discriminator training loss for a batch data.

Parameters

- **real_data** (*torch.LongTensor*) – Real data of the batch, shape: [batch_size, max_length]

- **fake_data** (*torch.LongTensor*) – Fake data of the batch, shape: [batch_size, max_length]

Returns
Training loss, shape: []

Return type
torch.Tensor

calculate_g_adversarial_loss()

Calculate the adversarial generator training loss for a batch data.

Returns
Training loss, shape: []

Return type
torch.Tensor

calculate_g_train_loss(*corpus*)

Calculate the generator training loss for a batch data.

Parameters
corpus (*Corpus*) – Corpus class of the batch.

Returns
Training loss, shape: []

Return type
torch.Tensor

calculate_nll_test(*eval_data*)

Calculate the negative log-likelihood of the batch.

Parameters
eval_data (*Corpus*) – Corpus class of the batch.

Returns
NLL_test of eval data

Return type
torch.FloatTensor

sample(*sample_num*)

Sample sample_num padded fake data generated by generator.

Parameters
sample_num (*int*) – The number of padded fake data generated by generator.

Returns
Fake data generated by generator, shape: [sample_num, max_length]

Return type
torch.LongTensor

training: bool

type = 2

class textbox.model.abstract_generator.**Seq2SeqGenerator**(*config*, *dataset*)

Bases: *AbstractModel*

This is a abstract general seq2seq generator. All the seq2seq model should implement this class. The base general seq2seq generator class provide the basic parameters information.

training: bool

type = 3

class textbox.model.abstract_generator.**UnconditionalGenerator**(*config*, *dataset*)

Bases: *AbstractModel*

This is a abstract general unconditional generator. All the unconditional model should implement this class. The base general unconditional generator class provide the basic parameters information.

training: bool

type = 1

4.2 textbox.model.init

textbox.model.init.**xavier_normal_initialization**(*module*)

using xavier_normal_ in PyTorch to initialize the parameters in nn.Embedding and nn.Linear layers. For bias in nn.Linear layers, using constant 0 to initialize.

Examples

```
>>> self.apply(xavier_normal_initialization)
```

textbox.model.init.**xavier_uniform_initialization**(*module*)

using xavier_uniform_ in PyTorch to initialize the parameters in nn.Embedding and nn.Linear layers. For bias in nn.Linear layers, using constant 0 to initialize.

Examples

```
>>> self.apply(xavier_uniform_initialization)
```

4.3 textbox.model.Attribute

4.3.1 Attr2Seq

Reference:
Li Dong et al. "Learning to Generate Product Reviews from Attributes" in 2017.

class textbox.model.Attribute.attr2seq.**Attr2Seq**(*config*, *dataset*)

Bases: *AttributeGenerator*

Attribute Encoder and RNN-based Decoder architecture is a basic frame work for Attr2Seq text generation.

encoder(*source_idx*)

Parameters
source_idx (*Torch.Tensor*) – source attribute index, shape: [batch_size, attribute_num].

Returns

- Torch.Tensor: output features, shape: [batch_size, attribute_num, embedding_size].
- Torch.Tensor: hidden states, shape: [num_dec_layers, batch_size, hidden_size].

Return type
tuple

forward(*corpus*, *epoch_idx=0*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

generate(*batch_data*, *eval_data*)

Predict the texts conditioned on a noise or sequence.

Parameters

- **batch_data** (*Corpus*) – Corpus class of a single batch.
- **eval_data** – Common data of all the batches.

Returns
Generated text, shape: [batch_size, max_len]

Return type
torch.Tensor

training: **bool**

4.3.2 C2S

Reference:
Jian Tang et al. "Context-aware Natural Language Generation with Recurrent Neural Networks" in 2016.

class textbox.model.Attribute.c2s.**C2S**(*config*, *dataset*)

Bases: *AttributeGenerator*

Context-aware Natural Language Generation with Recurrent Neural Network

encoder(*attr_data*)

forward(*corpus*, *epoch_idx=-1*, *nll_test=False*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

generate(*batch_data*, *eval_data*)

Predict the texts conditioned on a noise or sequence.

Parameters

- **batch_data** (*Corpus*) – Corpus class of a single batch.
- **eval_data** – Common data of all the batches.

Returns
Generated text, shape: [batch_size, max_len]

Return type
torch.Tensor

training: bool

4.4 textbox.model.GAN

4.4.1 TextGAN

Reference:
Zhang et al. "Adversarial Feature Matching for Text Generation" in ICML 2017.

class textbox.model.GAN.textgan.**TextGAN**(*config*, *dataset*)

Bases: *GenerativeAdversarialNet*

TextGAN is a generative adversarial network, which proposes matching the high-dimensional latent feature distributions of real and synthetic sentences, via a kernelized discrepancy metric.

calculate_d_train_loss(*real_data*, *fake_data*, *z*, *epoch_idx*)

Calculate the discriminator training loss for a batch data.

Parameters

- **real_data** (*torch.LongTensor*) – Real data of the batch, shape: [batch_size, max_length]
- **fake_data** (*torch.LongTensor*) – Fake data of the batch, shape: [batch_size, max_length]

Returns
Training loss, shape: []

Return type
torch.Tensor

calculate_g_adversarial_loss(*real_data*, *epoch_idx*)

Calculate the adversarial generator training loss for a batch data.

Returns
Training loss, shape: []

Return type
torch.Tensor

calculate_g_train_loss(*corpus*, *epoch_idx*)

Calculate the generator training loss for a batch data.

Parameters
corpus (*Corpus*) – Corpus class of the batch.

Returns
Training loss, shape: []

Return type
torch.Tensor

calculate_nll_test(*corpus*, *epoch_idx*)

Calculate the negative log-likelihood of the batch.

Parameters
eval_data (*Corpus*) – Corpus class of the batch.

Returns
NLL_test of eval data

Return type
torch.FloatTensor

generate(*batch_data*, *eval_data*)

Predict the texts conditioned on a noise or sequence.

Parameters

- **batch_data** (*Corpus*) – Corpus class of a single batch.
- **eval_data** – Common data of all the batches.

Returns
Generated text, shape: [batch_size, max_len]

Return type
torch.Tensor

sample()

Sample sample_num padded fake data generated by generator.

Parameters
sample_num (*int*) – The number of padded fake data generated by generator.

Returns
Fake data generated by generator, shape: [sample_num, max_length]

Return type
torch.LongTensor

training: bool

4.4.2 SeqGAN

Reference:
Yu et al. "SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient" in AAAI 2017.

class textbox.model.GAN.seqgan.**SeqGAN**(*config*, *dataset*)

Bases: *GenerativeAdversarialNet*

SeqGAN is a generative adversarial network consisting of a generator and a discriminator. Modeling the data generator as a stochastic policy in reinforcement learning (RL), SeqGAN bypasses the generator differentiation problem by directly performing gradient policy update. The RL reward signal comes from the GAN discriminator

judged on a complete sequence, and is passed back to the intermediate state-action steps using Monte Carlo search.

`calculate_d_train_loss`(*real_data*, *fake_data*, *epoch_idx*)

Calculate the discriminator training loss for a batch data.

Parameters

- **real_data** (*torch.LongTensor*) – Real data of the batch, shape: [batch_size, max_length]
- **fake_data** (*torch.LongTensor*) – Fake data of the batch, shape: [batch_size, max_length]

Returns
Training loss, shape: []

Return type
torch.Tensor

`calculate_g_adversarial_loss`(*epoch_idx*)

Calculate the adversarial generator training loss for a batch data.

Returns
Training loss, shape: []

Return type
torch.Tensor

`calculate_g_train_loss`(*corpus*, *epoch_idx*)

Calculate the generator training loss for a batch data.

Parameters
corpus (*Corpus*) – Corpus class of the batch.

Returns
Training loss, shape: []

Return type
torch.Tensor

`calculate_nll_test`(*corpus*, *epoch_idx*)

Calculate the negative log-likelihood of the batch.

Parameters
eval_data (*Corpus*) – Corpus class of the batch.

Returns
NLL_test of eval data

Return type
torch.FloatTensor

`generate`(*batch_data*, *eval_data*)

Predict the texts conditioned on a noise or sequence.

Parameters

- **batch_data** (*Corpus*) – Corpus class of a single batch.
- **eval_data** – Common data of all the batches.

Returns
Generated text, shape: [batch_size, max_len]

Return type
torch.Tensor

sample(*sample_num*)

Sample sample_num padded fake data generated by generator.

Parameters
sample_num (*int*) – The number of padded fake data generated by generator.

Returns
Fake data generated by generator, shape: [sample_num, max_length]

Return type
torch.LongTensor

training: **bool**

4.4.3 RankGAN

Reference:
Lin et al. "Adversarial Ranking for Language Generation" in NIPS 2017.

class textbox.model.GAN.rankgan.**RankGAN**(*config*, *dataset*)

Bases: *GenerativeAdversarialNet*

RankGAN is a generative adversarial network consisting of a generator and a ranker. The ranker is trained to rank the machine-written sentences lower than human-written sentences with respect to reference sentences. The generator is trained to synthesize sentences that can be ranked higher than the human-written one. We implement the model following the original author.

calculate_d_train_loss(*real_data*, *fake_data*, *ref_data*, *epoch_idx*)

Calculate the discriminator training loss for a batch data.

Parameters

- **real_data** (*torch.LongTensor*) – Real data of the batch, shape: [batch_size, max_length]
- **fake_data** (*torch.LongTensor*) – Fake data of the batch, shape: [batch_size, max_length]

Returns
Training loss, shape: []

Return type
torch.Tensor

calculate_g_adversarial_loss(*ref_data*, *epoch_idx*)

Calculate the adversarial generator training loss for a batch data.

Returns
Training loss, shape: []

Return type
torch.Tensor

calculate_g_train_loss(*corpus*, *epoch_idx*)

Calculate the generator training loss for a batch data.

Parameters
corpus (*Corpus*) – Corpus class of the batch.

Returns
Training loss, shape: []

Return type
torch.Tensor

calculate_nll_test(*corpus*, *epoch_idx*)

Calculate the negative log-likelihood of the batch.

Parameters
eval_data (*Corpus*) – Corpus class of the batch.

Returns
NLL_test of eval data

Return type
torch.FloatTensor

generate(*batch_data*, *eval_data*)

Predict the texts conditioned on a noise or sequence.

Parameters

- **batch_data** (*Corpus*) – Corpus class of a single batch.
- **eval_data** – Common data of all the batches.

Returns
Generated text, shape: [batch_size, max_len]

Return type
torch.Tensor

sample(*sample_num*)

Sample sample_num padded fake data generated by generator.

Parameters
sample_num (*int*) – The number of padded fake data generated by generator.

Returns
Fake data generated by generator, shape: [sample_num, max_length]

Return type
torch.LongTensor

training: bool

4.4.4 MaliGAN

Reference:
Che et al. "Maximum-Likelihood Augmented Discrete Generative Adversarial Networks".

class textbox.model.GAN.maligan.**MaliGAN**(*config*, *dataset*)

Bases: *GenerativeAdversarialNet*

MaliGAN is a generative adversarial network using a normalized maximum likelihood optimization.

calculate_d_train_loss(*real_data*, *fake_data*, *epoch_idx*)

Calculate the discriminator training loss for a batch data.

Parameters

- **real_data** (*torch.LongTensor*) – Real data of the batch, shape: [batch_size, max_length]
- **fake_data** (*torch.LongTensor*) – Fake data of the batch, shape: [batch_size, max_length]

Returns
Training loss, shape: []

Return type
torch.Tensor

calculate_g_adversarial_loss(*epoch_idx*)

Calculate the adversarial generator training loss for a batch data.

Returns
Training loss, shape: []

Return type
torch.Tensor

calculate_g_train_loss(*corpus*, *epoch_idx*)

Calculate the generator training loss for a batch data.

Parameters
corpus (*Corpus*) – Corpus class of the batch.

Returns
Training loss, shape: []

Return type
torch.Tensor

calculate_nll_test(*corpus*, *epoch_idx*)

Calculate the negative log-likelihood of the batch.

Parameters
eval_data (*Corpus*) – Corpus class of the batch.

Returns
NLL_test of eval data

Return type
torch.FloatTensor

generate(*batch_data*, *eval_data*)

Predict the texts conditioned on a noise or sequence.

Parameters

- **batch_data** (*Corpus*) – Corpus class of a single batch.
- **eval_data** – Common data of all the batches.

Returns
Generated text, shape: [batch_size, max_len]

Return type
torch.Tensor

sample(*sample_num*)

Sample sample_num padded fake data generated by generator.

Parameters
sample_num (*int*) – The number of padded fake data generated by generator.

Returns
Fake data generated by generator, shape: [sample_num, max_length]

Return type
torch.LongTensor

training: **bool**

4.4.5 LeakGAN

Reference:
Guo et al. "Long Text Generation via Adversarial Training with Leaked Information" in AAAI 2018.

class textbox.model.GAN.leakgan.**LeakGAN**(*config*, *dataset*)

Bases: *GenerativeAdversarialNet*

LeakGAN is a generative adversarial network to address the problem for long text generation. We allow the discriminative net to leak its own high-level extracted features to the generative net to further help the guidance. The generator incorporates such informative signals into all generation steps through an additional Manager module, which takes the extracted features of current generated words and outputs a latent vector to guide the Worker module for next-word generation.

calculate_d_train_loss(*real_data*, *fake_data*, *epoch_idx*)

Calculate the discriminator training loss for a batch data.

Parameters

- **real_data** (*torch.LongTensor*) – Real data of the batch, shape: [batch_size, max_length]
- **fake_data** (*torch.LongTensor*) – Fake data of the batch, shape: [batch_size, max_length]

Returns
Training loss, shape: []

Return type
torch.Tensor

calculate_g_adversarial_loss(*epoch_idx*)

Calculate the adversarial generator training loss for a batch data.

Returns
Training loss, shape: []

Return type
torch.Tensor

calculate_g_train_loss(*corpus*, *epoch_idx*)

Calculate the generator training loss for a batch data.

Parameters
corpus (*Corpus*) – Corpus class of the batch.

Returns
Training loss, shape: []

Return type
torch.Tensor

`calculate_nll_test`(*corpus*, *epoch_idx*)

Calculate the negative log-likelihood of the batch.

Parameters
eval_data (*Corpus*) – Corpus class of the batch.

Returns
NLL_test of eval data

Return type
torch.FloatTensor

`generate`(*batch_data*, *eval_data*)

Predict the texts conditioned on a noise or sequence.

Parameters

- **batch_data** (*Corpus*) – Corpus class of a single batch.
- **eval_data** – Common data of all the batches.

Returns
Generated text, shape: [batch_size, max_len]

Return type
torch.Tensor

`sample`(*sample_num*)

Sample sample_num padded fake data generated by generator.

Parameters
sample_num (*int*) – The number of padded fake data generated by generator.

Returns
Fake data generated by generator, shape: [sample_num, max_length]

Return type
torch.LongTensor

`training: bool`

4.4.6 MaskGAN

Reference:
Fedus et al. "MaskGAN: Better Text Generation via Filling in the _____" in ICLR 2018.

`class textbox.model.GAN.maskgan.MaskGAN`(*config*, *dataset*)

Bases: *GenerativeAdversarialNet*

MaskGAN is a generative adversarial network to improve sample quality, which introduces an actor-critic conditional GAN that fills in missing text conditioned on the surrounding context.

`calculate_d_train_loss`(*data*, *epoch_idx*)

Specified for maskgan calculate discriminator masked token predicted

`calculate_g_adversarial_loss`(*data*, *epoch_idx*)

Specified for maskgan calculate adversarial masked token predicted

calculate_g_train_loss(*corpus*, *epoch_idx=0*, *validate=False*)

Specified for maskgan calculate generator masked token predicted

calculate_nll_test(*eval_batch*, *epoch_idx*)

Specified for maskgan calculating the negative log-likelihood of the batch.

generate(*batch_data*, *eval_data*)

Predict the texts conditioned on a noise or sequence.

Parameters

- **batch_data** (*Corpus*) – Corpus class of a single batch.
- **eval_data** – Common data of all the batches.

Returns
Generated text, shape: [batch_size, max_len]

Return type
torch.Tensor

generate_mask(*batch_size*, *seq_len*, *mask_strategy*)

Generate the mask to be fed into the model.

training: bool

update_is_present_rate()

4.5 textbox.model.LM

4.5.1 RNN

class textbox.model.LM.rnn.**RNN**(*config*, *dataset*)

Bases: *UnconditionalGenerator*

Basic Recurrent Neural Network for Maximum Likelihood Estimation.

calculate_nll_test(*corpus*, *epoch_idx*)

forward(*corpus*, *epoch_idx=- 1*, *nll_test=False*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

generate(*batch_data*, *eval_data*)

Predict the texts conditioned on a noise or sequence.

Parameters

- **batch_data** (*Corpus*) – Corpus class of a single batch.
- **eval_data** – Common data of all the batches.

Returns
Generated text, shape: [batch_size, max_len]

Return type
torch.Tensor

training: **bool**

4.5.2 GPT-2

Reference:
Radford et al. "Language models are unsupervised multitask".

class textbox.model.LM.gpt2.**GPT2**(*config*, *dataset*)

Bases: *UnconditionalGenerator*

GPT-2 is an auto-regressive language model with stacked Transformer decoders.

calculate_nll_test(*corpus*, *epoch_idx=- 1*)

forward(*corpus*, *epoch_idx=- 1*, *nll_test=False*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

generate(*batch_data*, *eval_data*)

Predict the texts conditioned on a noise or sequence.

Parameters

- **batch_data** (*Corpus*) – Corpus class of a single batch.
- **eval_data** – Common data of all the batches.

Returns
Generated text, shape: [batch_size, max_len]

Return type
torch.Tensor

training: **bool**

4.5.3 XLNet

Reference:
Yang et al. "XLNet: Generalized Autoregressive Pretraining for Language Understanding" in NIPS 2019.

class textbox.model.LM.xlnet.**XLNet**(*config*, *dataset*)

Bases: *UnconditionalGenerator*

XLnet is an extension of the Transformer-XL model pre-trained using an autoregressive method to learn bidirectional contexts by maximizing the expected likelihood over all permutations of the input sequence factorization order.

calculate_nll_test(*corpus*, *epoch_idx=- 1*)

forward(*corpus*, *epoch_idx=- 1*, *nll_test=False*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

generate(*batch_data*, *eval_data*)

Predict the texts conditioned on a noise or sequence.

Parameters

- **batch_data** (*Corpus*) – Corpus class of a single batch.
- **eval_data** – Common data of all the batches.

Returns
Generated text, shape: [batch_size, max_len]

Return type
torch.Tensor

training: **bool**

4.6 textbox.model.Seq2Seq

4.6.1 RNNEncDec

Reference:
Sutskever et al. "Sequence to Sequence Learning with Neural Networks" in NIPS 2014.

class textbox.model.Seq2Seq.rnnencdec.**RNNEncDec**(*config*, *dataset*)

Bases: *Seq2SeqGenerator*

RNN-based Encoder-Decoder architecture is a basic framework for Seq2Seq text generation.

forward(*corpus*, *epoch_idx=0*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

generate(*batch_data*, *eval_data*)

Predict the texts conditioned on a noise or sequence.

Parameters

- **batch_data** (*Corpus*) – Corpus class of a single batch.

- **eval_data** – Common data of all the batches.

Returns
Generated text, shape: [batch_size, max_len]

Return type
torch.Tensor

training: **bool**

4.6.2 TransformerEncDec

Reference:
Vaswani et al. "Attention is All you Need" in NIPS 2017.

class textbox.model.Seq2Seq.transformerencdec.**TransformerEncDec**(*config*, *dataset*)

Bases: *Seq2SeqGenerator*

Transformer-based Encoder-Decoder architecture is a powerful framework for Seq2Seq text generation.

forward(*corpus*, *epoch_idx=0*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

generate(*batch_data*, *eval_data*)

Predict the texts conditioned on a noise or sequence.

Parameters

- **batch_data** (*Corpus*) – Corpus class of a single batch.
- **eval_data** – Common data of all the batches.

Returns
Generated text, shape: [batch_size, max_len]

Return type
torch.Tensor

reset_parameters()

training: **bool**

4.6.3 HierarchicalRNN

Reference:
Serban et al. "Building End-To-End Dialogue Systems Using Generative Hierarchical Neural Network Models" in AAAI 2016.

class textbox.model.Seq2Seq.hred.**HRED**(*config*, *dataset*)

Bases: *Seq2SeqGenerator*

This is a description

encode(*corpus*)

forward(*corpus*, *epoch_idx=0*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

generate(*batch_data*, *eval_data*)

Predict the texts conditioned on a noise or sequence.

Parameters

- **batch_data** (*Corpus*) – Corpus class of a single batch.
- **eval_data** – Common data of all the batches.

Returns
Generated text, shape: [batch_size, max_len]

Return type
torch.Tensor

training: **bool**

4.7 textbox.model.VAE

4.7.1 RNNVAE

Reference:
Bowman et al. "Generating Sentences from a Continuous Space" in CoNLL 2016.

class textbox.model.VAE.rnnvae.**RNNVAE**(*config*, *dataset*)

Bases: *UnconditionalGenerator*

LSTMVAE is the first text generation model with VAE, we modify its architecture to fit all RNN type, and rename it as RNNVAE

calculate_nll_test(*corpus*, *epoch_idx=0*)

forward(*corpus*, *epoch_idx=0*, *nll_test=False*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

generate(*batch_data*, *eval_data*)

Predict the texts conditioned on a noise or sequence.

Parameters

- **batch_data** (*Corpus*) – Corpus class of a single batch.
- **eval_data** – Common data of all the batches.

Returns
Generated text, shape: [batch_size, max_len]

Return type
torch.Tensor

training: **bool**

4.7.2 CNNVAE

Reference:
Yang et al. "Improved Variational Autoencoders for Text Modeling using Dilated Convolutions" in ICML 2017.

class textbox.model.VAE.cnnvae.**CNNVAE**(*config*, *dataset*)

Bases: *UnconditionalGenerator*

CNNVAE use a dilated CNN as decoder, which made a trade-off between contextual capacity of the decoder and effective use of encoding information.

calculate_nll_test(*corpus*, *epoch_idx=0*)

forward(*corpus*, *epoch_idx=0*, *nll_test=False*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

generate(*batch_data*, *eval_data*)

Predict the texts conditioned on a noise or sequence.

Parameters

- **batch_data** (*Corpus*) – Corpus class of a single batch.
- **eval_data** – Common data of all the batches.

Returns
Generated text, shape: [batch_size, max_len]

Return type
torch.Tensor

training: **bool**

4.7.3 HybridVAE

Reference:
Rothe et al. "A Hybrid Convolutional Variational Autoencoder for Text Generation" in EMNLP 2017.

class textbox.model.VAE.hybridvae.**HybridVAE**(*config*, *dataset*)

Bases: *UnconditionalGenerator*

HybridVAE blends fully feed-forward convolutional and deconvolutional components with a recurrent language model.

calculate_nll_test(*corpus*, *epoch_idx=0*)

forward(*corpus*, *epoch_idx=0*, *nll_test=False*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

generate(*batch_data*, *eval_data*)

Predict the texts conditioned on a noise or sequence.

Parameters

- **batch_data** (*Corpus*) – Corpus class of a single batch.
- **eval_data** – Common data of all the batches.

Returns
Generated text, shape: [batch_size, max_len]

Return type
torch.Tensor

training: **bool**

4.7.4 Conditional VAE

Reference:
Juntao Li et al. "Generating Classical Chinese Poems via Conditional Variational Autoencoder and Adversarial Training" in ACL 2018.

class textbox.model.VAE.cvae.**CVAE**(*config*, *dataset*)

Bases: *Seq2SeqGenerator*

We use the title of a poem and the previous line as condition to generate the current line.

forward(*corpus*, *epoch_idx=0*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

generate(*batch_data*, *eval_data*)

Predict the texts conditioned on a noise or sequence.

Parameters

- **batch_data** (*Corpus*) – Corpus class of a single batch.
- **eval_data** – Common data of all the batches.

Returns
Generated text, shape: [batch_size, max_len]

Return type
torch.Tensor

training: bool

xavier_uniform_initialization(*module*)

using uniform in PyTorch to initialize the parameters in nn.Embedding and nn.Linear layers. For bias in nn.Linear layers, using constant 0 to initialize.

CHAPTER

FIVE

TEXTBOX.MODULE

5.1 textbox.module.layers

Common Layers in text generation

class textbox.module.layers.**Highway**(*num_highway_layers*, *input_size*)

Bases: Module

Highway Layers

Parameters

- **num_highway_layers** (-) – number of highway layers.
- **input_size** (-) – size of highway input.

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

training: bool

class textbox.module.layers.**TransformerLayer**(*embedding_size*, *ffn_size*, *num_heads*, *attn_dropout_ratio=0.0*, *attn_weight_dropout_ratio=0.0*, *ffn_dropout_ratio=0.0*, *with_external=False*)

Bases: Module

Transformer Layer, including
a multi-head self-attention, a external multi-head self-attention layer (only for conditional decoder) and a point-wise feed-forward layer.

Parameters

- **self_padding_mask** (*torch.bool*) – the padding mask for the multi head attention sublayer.
- **self_attn_mask** (*torch.bool*) – the attention mask for the multi head attention sublayer.
- **external_states** (*torch.Tensor*) – the external context for decoder, e.g., hidden states from encoder.

- **external_padding_mask** (*torch.bool*) – the padding mask for the external states.

Returns
the output of the point-wise feed-forward sublayer, is the output of the transformer layer

Return type
feedforward_output (torch.Tensor)

forward(*x*, *kv=None*, *self_padding_mask=None*, *self_attn_mask=None*, *external_states=None*, *external_padding_mask=None*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

gelu(*x*)

reset_parameters()

training: bool

5.2 textbox.module.strategy

Common Strategys in text generation

class textbox.module.strategy.**Beam_Search_Hypothesis**(*beam_size*, *sos_token_idx*, *eos_token_idx*, *device*, *idx2token*)

Bases: `object`

Class designed for beam search.

generate()

Pick the hypothesis with max prob among beam_size hypothesises.

Returns
the generated tokens

Return type
List[str]

step(*gen_idx*, *token_logits*, *decoder_states=None*, *encoder_output=None*, *encoder_mask=None*, *input_type='token'*)

A step for beam search.

Parameters

- **gen_idx** (*int*) – the generated step number.
- **token_logits** (*torch.Tensor*) – logits distribution, shape: [hyp_num, sequence_length, vocab_size].
- **decoder_states** (*torch.Tensor, optional*) – the states of decoder needed to choose, shape: [hyp_num, sequence_length, hidden_size], default: None.

- **encoder_output** (*torch.Tensor, optional*) – the output of encoder needed to copy, shape: [hyp_num, sequence_length, hidden_size], default: None.
- **encoder_mask** (*torch.Tensor, optional*) – the mask of encoder to copy, shape: [hyp_num, sequence_length], default: None.

Returns
the next input squence, shape: [hyp_num], torch.Tensor, optional: the chosen states of decoder, shape: [new_hyp_num, sequence_length, hidden_size] torch.Tensor, optional: the copyed output of encoder, shape: [new_hyp_num, sequence_length, hidden_size] torch.Tensor, optional: the copyed mask of encoder, shape: [new_hyp_num, sequence_length]

Return type
torch.Tensor

stop()

Determine if the beam search is over.

Returns
`True` represents the search over, *Flase* represents the search working.

Return type
Bool

class textbox.module.strategy.**Copy_Beam_Search**(*beam_size*, *sos_token_idx*, *eos_token_idx*, *unknown_token_idx*, *device*, *idx2token*, *is_attention=False*, *is_pgen=False*, *is_coverage=False*)

Bases: `object`

generate()

step(*gen_idx*, *vocab_dists*, *decoder_hidden_states*, *kwargs=None*)

stop()

textbox.module.strategy.**greedy_search**(*logits*)

Find the index of max logits

Parameters
logits (*torch.Tensor*) – logits distribution

Returns
the chosen index of token

Return type
torch.Tensor

textbox.module.strategy.**topk_sampling**(*logits*, *temperature=1.0*, *top_k=0*, *top_p=0.9*)

Filter a distribution of logits using top-k and/or nucleus (top-p) filtering

Parameters

- **logits** (*torch.Tensor*) – logits distribution
- **>0** (*top_k*) – keep only top k tokens with highest probability (top-k filtering).
- **>0.0** (*top_p*) – keep the top tokens with cumulative probability >= top_p (nucleus filtering).

Returns
the chosen index of token.

Return type
torch.Tensor

5.3 textbox.module.Attention

5.3.1 Attention Layers

class textbox.module.Attention.attention_mechanism.**BahdanauAttention**(*source_size*, *target_size*)

Bases: Module

Bahdanau Attention is proposed in the following paper:
Neural Machine Translation by Jointly Learning to Align and Translate.

Reference:
https://arxiv.org/abs/1409.0473

forward(*hidden_states*, *encoder_outputs*, *encoder_masks*)

Bahdanau attention

Parameters

- **hidden_states** – shape: [batch_size, tgt_len, target_size]
- **encoder_outputs** – shape: [batch_size, src_len, source_size]
- **encoder_masks** – shape: [batch_size, src_len]

Returns

- context: shape: [batch_size, tgt_len, source_size]
- probs: shape: [batch_size, tgt_len, src_len]

Return type
tuple

score(*hidden_states*, *encoder_outputs*)

Calculate the attention scores between encoder outputs and decoder states.

training: bool

class textbox.module.Attention.attention_mechanism.**LuongAttention**(*source_size*, *target_size*, *alignment_method='concat'*, *is_coverage=False*)

Bases: Module

Luong Attention is proposed in the following paper: Effective Approaches to Attention-based Neural Machine Translation.

Reference:
https://arxiv.org/abs/1508.04025

forward(*hidden_states*, *encoder_outputs*, *encoder_masks*, *coverages=None*)

Luong attention

Parameters

- **hidden_states** – shape: [batch_size, tgt_len, target_size]
- **encoder_outputs** – shape: [batch_size, src_len, source_size]

- **encoder_masks** – shape: [batch_size, src_len]

Returns

- context: shape: [batch_size, tgt_len, source_size]
- probs: shape: [batch_size, tgt_len, src_len]

Return type
tuple

score(*hidden_states*, *encoder_outputs*, *coverages=None*)

Calculate the attention scores between encoder outputs and decoder states.

training: bool

class textbox.module.Attention.attention_mechanism.**MonotonicAttention**(*source_size*, *target_size*, *init_r=- 4*)

Bases: Module

Monotonic Attention is proposed in the following paper:
Online and Linear-Time Attention by Enforcing Monotonic Alignments.

Reference:
https://arxiv.org/abs/1704.00784

exclusive_cumprod(*x*)

Exclusive cumulative product [a, b, c] => [1, a, a * b]

gaussian_noise(**size*)

Additive gaussian nosie to encourage discreteness

hard(*hidden_states*, *encoder_outputs*, *encoder_masks*, *previous_probs=None*)

Hard monotonic attention (Test)

Parameters

- **hidden_states** – shape: [batch_size, tgt_len, target_size]
- **encoder_outputs** – shape: [batch_size, src_len, source_size]
- **encoder_masks** – shape: [batch_size, src_len]
- **previous_probs** – shape: [batch_size, tgt_len, src_len]

Returns

- context: shape: [batch_size, tgt_len, source_size]
- probs: shape: [batch_size, tgt_len, src_len]

Return type
tuple

safe_cumprod(*x*)

Numerically stable cumulative product by cumulative sum in log-space

score(*hidden_states*, *encoder_outputs*)

Calculate the attention scores between encoder outputs and decoder states.

soft(*hidden_states*, *encoder_outputs*, *encoder_masks*, *previous_probs=None*)

Soft monotonic attention (Train)

Parameters

- **hidden_states** – shape: [batch_size, tgt_len, target_size]
- **encoder_outputs** – shape: [batch_size, src_len, source_size]
- **encoder_masks** – shape: [batch_size, src_len]
- **previous_probs** – shape: [batch_size, tgt_len, src_len]

Returns

- context: shape: [batch_size, tgt_len, source_size]
- probs: shape: [batch_size, tgt_len, src_len]

Return type
tuple

training: bool

class textbox.module.Attention.attention_mechanism.**MultiHeadAttention**(*embedding_size*, *num_heads*, *attn_weight_dropout_ratio=0.0*, *return_distribute=False*)

Bases: Module

Multi-head Attention is proposed in the following paper:
Attention Is All You Need.

Reference:
https://arxiv.org/abs/1706.03762

forward(*query*, *key*, *value*, *key_padding_mask=None*, *attn_mask=None*)

Multi-head attention

Parameters

- **query** – shape: [batch_size, tgt_len, embedding_size]
- **value** (*key and*) – shape: [batch_size, src_len, embedding_size]
- **key_padding_mask** – shape: [batch_size, src_len]
- **attn_mask** – shape: [batch_size, tgt_len, src_len]

Returns

- attn_repre: shape: [batch_size, tgt_len, embedding_size]
- attn_weights: shape: [batch_size, tgt_len, src_len]

Return type
tuple

reset_parameters()

training: bool

class textbox.module.Attention.attention_mechanism.**SelfAttentionMask**(*init_size=100*)

Bases: Module

forward(*size*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

static get_mask(*size*)

training: bool

5.4 textbox.module.Decoder

5.4.1 CNN Decoder

class textbox.module.Decoder.cnn_decoder.**BasicCNNDecoder**(*input_size*, *latent_size*, *decoder_kernel_size*, *decoder_dilations*, *dropout_ratio*)

Bases: `Module`

Basic Convolution Neural Network (CNN) decoder. Code Reference: https://github.com/kefirski/contiguous-succotash

forward(*decoder_input*, *noise*)

Implement the decoding process.

Parameters

- **decoder_input** (*Torch.Tensor*) – target sequence embedding, shape: [batch_size, sequence_length, embedding_size].
- **noise** (*Torch.Tensor*) – latent code, shape: [batch_size, latent_size].

Returns
output features, shape: [batch_size, sequence_length, feature_size].

Return type
torch.Tensor

training: bool

class textbox.module.Decoder.cnn_decoder.**HybridDecoder**(*embedding_size*, *latent_size*, *hidden_size*, *num_dec_layers*, *rnn_type*, *vocab_size*)

Bases: `Module`

Hybrid Convolution Neural Network (CNN) and Recurrent Neural Network (RNN) decoder. Code Reference: https://github.com/kefirski/hybrid_rvae

conv_decoder(*latent_variable*)

Implement the CNN decoder.

Parameters
latent_variable (*Torch.Tensor*) – latent code, shape: [batch_size, latent_size].

Returns
output features, shape: [batch_size, sequence_length, feature_size].

Return type
torch.Tensor

forward(*decoder_input*, *latent_variable*)

Implement the decoding process.

Parameters

- **decoder_input** (*Torch.Tensor*) – target sequence embedding, shape: [batch_size, sequence_length, embedding_size].
- **latent_variable** (*Torch.Tensor*) – latent code, shape: [batch_size, latent_size].

Returns

- torch.Tensor: RNN output features, shape: [batch_size, sequence_length, feature_size].
- torch.Tensor: CNN output features, shape: [batch_size, sequence_length, feature_size].

Return type
tuple

rnn_decoder(*cnn_logits*, *decoder_input*, *initial_state=None*)

Implement the RNN decoder using CNN output.

Parameters

- **cnn_logits** (*Torch.Tensor*) – latent code, shape: [batch_size, sequence_length, feature_size].
- **decoder_input** (*Torch.Tensor*) – target sequence embedding, shape: [batch_size, sequence_length, embedding_size].
- **initial_state** (*Torch.Tensor*) – initial hidden states, default: None.

Returns

- Torch.Tensor: output features, shape: [batch_size, sequence_length, num_directions * hidden_size].
- Torch.Tensor: hidden states, shape: [batch_size, num_layers * num_directions, hidden_size].

Return type
tuple

training: **bool**

5.4.2 RNN Decoder

class textbox.module.Decoder.rnn_decoder.**AttentionalRNNDecoder**(*embedding_size*, *hidden_size*, *context_size*, *num_dec_layers*, *rnn_type*, *dropout_ratio=0.0*, *attention_type='LuongAttention'*, *alignment_method='concat'*)

Bases: Module

Attention-based Recurrent Neural Network (RNN) decoder.

forward(*input_embeddings*, *hidden_states=None*, *encoder_outputs=None*, *encoder_masks=None*, *previous_probs=None*)

Implement the attention-based decoding process.

Parameters

- **input_embeddings** (*Torch.Tensor*) – source sequence embedding, shape: [batch_size, sequence_length, embedding_size].
- **hidden_states** (*Torch.Tensor*) – initial hidden states, default: None.
- **encoder_outputs** (*Torch.Tensor*) – encoder output features, shape: [batch_size, sequence_length, hidden_size], default: None.
- **encoder_masks** (*Torch.Tensor*) – encoder state masks, shape: [batch_size, sequence_length], default: None.

Returns

- Torch.Tensor: output features, shape: [batch_size, sequence_length, num_directions * hidden_size].
- Torch.Tensor: hidden states, shape: [batch_size, num_layers * num_directions, hidden_size].

Return type
tuple

init_hidden(*input_embeddings*)

Initialize initial hidden states of RNN.

Parameters
input_embeddings (*Torch.Tensor*) – input sequence embedding, shape: [batch_size, sequence_length, embedding_size].

Returns
the initial hidden states.

Return type
Torch.Tensor

training: **bool**

class textbox.module.Decoder.rnn_decoder.**BasicRNNDecoder**(*embedding_size*, *hidden_size*, *num_dec_layers*, *rnn_type*, *dropout_ratio=0.0*)

Bases: Module

Basic Recurrent Neural Network (RNN) decoder.

forward(*input_embeddings*, *hidden_states=None*)

Implement the decoding process.

Parameters

- **input_embeddings** (*Torch.Tensor*) – target sequence embedding, shape: [batch_size, sequence_length, embedding_size].
- **hidden_states** (*Torch.Tensor*) – initial hidden states, default: None.

Returns

- Torch.Tensor: output features, shape: [batch_size, sequence_length, num_directions * hidden_size].
- Torch.Tensor: hidden states, shape: [num_layers * num_directions, batch_size, hidden_size].

Return type
tuple

init_hidden(*input_embeddings*)

Initialize initial hidden states of RNN.

Parameters
input_embeddings (*Torch.Tensor*) – input sequence embedding, shape: [batch_size, sequence_length, embedding_size].

Returns
the initial hidden states.

Return type
Torch.Tensor

training: bool

class textbox.module.Decoder.rnn_decoder.**PointerRNNDecoder**(*vocab_size*, *embedding_size*, *hidden_size*, *context_size*, *num_dec_layers*, *rnn_type*, *dropout_ratio=0.0*, *is_attention=False*, *is_pgen=False*, *is_coverage=False*)

Bases: Module

forward(*input_embeddings*, *decoder_hidden_states*, *kwargs=None*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

training: bool

5.4.3 Transformer Decoder

class textbox.module.Decoder.transformer_decoder.**TransformerDecoder**(*embedding_size*, *ffn_size*, *num_dec_layers*, *num_heads*, *attn_dropout_ratio=0.0*, *attn_weight_dropout_ratio=0.0*, *ffn_dropout_ratio=0.0*, *with_external=True*)

Bases: Module

The stacked Transformer decoder layers.

forward(*x*, *kv=None*, *self_padding_mask=None*, *self_attn_mask=None*, *external_states=None*, *external_padding_mask=None*)

Implement the decoding process step by step.

Parameters

- **x** (*Torch.Tensor*) – target sequence embedding, shape: [batch_size, sequence_length, embedding_size].

- **kv** (*Torch.Tensor*) – the cached history latent vector, shape: [batch_size, sequence_length, embedding_size], default: None.
- **self_padding_mask** (*Torch.Tensor*) – padding mask of target sequence, shape: [batch_size, sequence_length], default: None.
- **self_attn_mask** (*Torch.Tensor*) – diagonal attention mask matrix of target sequence, shape: [batch_size, sequence_length, sequence_length], default: None.
- **external_states** (*Torch.Tensor*) – output features of encoder, shape: [batch_size, sequence_length, feature_size], default: None.
- **external_padding_mask** (*Torch.Tensor*) – padding mask of source sequence, shape: [batch_size, sequence_length], default: None.

Returns
output features, shape: [batch_size, sequence_length, ffn_size].

Return type
Torch.Tensor

training: **bool**

5.5 textbox.module.Discriminator

5.5.1 TextGAN Discriminator

class textbox.module.Discriminator.TextGANDiscriminator.**TextGANDiscriminator**(*config*, *dataset*)

Bases: *UnconditionalGenerator*

The discriminator of TextGAN.

calculate_g_loss(*real_data*, *fake_data*)

Calculate the maximum mean discrepancy loss for real data and fake data.

Parameters

- **real_data** (*torch.Tensor*) – The realistic sentence data, shape: [batch_size, max_seq_len].
- **fake_data** (*torch.Tensor*) – The generated sentence data, shape: [batch_size, max_seq_len].

Returns
The calculated mmd loss of real data and fake data, shape: [].

Return type
torch.Tensor

calculate_loss(*real_data*, *fake_data*, *z*)

Calculate the loss for real data and fake data.

Parameters

- **real_data** (*torch.Tensor*) – The realistic sentence data, shape: [batch_size, max_seq_len].
- **fake_data** (*torch.Tensor*) – The generated sentence data, shape: [batch_size, max_seq_len].

- **z** (*torch.Tensor*) – The latent code for generation, shape: [batch_size, hidden_size].

Returns
The calculated loss of real data and fake data, shape: [].

Return type
torch.Tensor

feature(*data*)

Get the feature map extracted from CNN for data.

Parameters
data (*torch.Tensor*) – The data to be extraced, shape: [batch_size, max_seq_len, vocab_size].

Returns
The feature of data, shape: [batch_size, total_filter_num].

Return type
torch.Tensor

forward(*data*)

Calculate the probability that the data is realistic.

Parameters
data (*torch.Tensor*) – The sentence data, shape: [batch_size, max_seq_len, vocab_size].

Returns
The probability that each sentence is realistic, shape: [batch_size].

Return type
torch.Tensor

training: **bool**

5.5.2 SeqGAN Discriminator

class textbox.module.Discriminator.SeqGANDiscriminator.**SeqGANDiscriminator**(*config*, *dataset*)

Bases: *UnconditionalGenerator*

The discriminator of SeqGAN.

calculate_loss(*real_data*, *fake_data*)

Calculate the loss for real data and fake data.

Parameters

- **real_data** (*torch.Tensor*) – The realistic sentence data, shape: [batch_size, max_seq_len].
- **fake_data** (*torch.Tensor*) – The generated sentence data, shape: [batch_size, max_seq_len].

Returns
The calculated loss of real data and fake data, shape: [].

Return type
torch.Tensor

forward(*data*)

Calculate the probability that the data is realistic.

Parameters
data (*torch.Tensor*) – The sentence data, shape: [batch_size, max_seq_len].

Returns
The probability that each sentence is realistic, shape: [batch_size].

Return type
torch.Tensor

training: **bool**

5.5.3 RankGAN Discriminator

class textbox.module.Discriminator.RankGANDiscriminator.**RankGANDiscriminator**(*config*, *dataset*)

Bases: *UnconditionalGenerator*

RankGANDiscriminator is a ranker which can endow a relative rank among the sequences when given a reference. The ranker is designed with the convolutional neural network.

calculate_loss(*real_data*, *fake_data*, *ref_data*)

Calculate the loss for real data and fake data. To rank the human_written sentences higher than the machine-written sentences.

Parameters

- **real_data** (*torch.Tensor*) – The realistic sentence data, shape: [batch_size, max_seq_len].
- **fake_data** (*torch.Tensor*) – The generated sentence data, shape: [batch_size, max_seq_len].
- **ref_data** (*torch.Tensor*) – The reference sentence data, shape: [ref_size, max_seq_len].

Returns
The calculated loss of real data and fake data, shape: [].

Return type
torch.Tensor

forward(*data*)

Maps concatenated sequence matrices into the embedded feature vectors.

Parameters
data (*torch.Tensor*) – The sentence data, shape: [batch_size, max_seq_len].

Returns
The embedded feature vectors, shape: [batch_size, total_filter_num].

Return type
torch.Tensor

get_rank_scores(*sample_data*, *ref_data*)

Get the ranking score (before softmax) for sample s given reference u.

$$\alpha(s|u) = cosine(y_s, y_u) = \frac{y_s \cdot y_u}{\| y_s \| \| y_u \|}$$

Parameters

- **sample_data** (*torch.Tensor*) – The realistic or generated sentence data, shape: [sample_size, max_seq_len].
- **ref_data** (*torch.Tensor*) – The reference sentence data, shape: [ref_size, max_seq_len].

Returns
The ranking score of sample data, shape: [batch_size].

Return type
torch.Tensor

highway(*data*)

Apply the highway net to data.

Parameters
data (*torch.Tensor*) – The original data, shape: [batch_size, total_filter_num].

Returns
The data processed after highway net, shape: [batch_size, total_filter_num].

Return type
torch.Tensor

training: **bool**

5.5.4 MaliGAN Discriminator

class textbox.module.Discriminator.MaliGANDiscriminator.**MaliGANDiscriminator**(*config*, *dataset*)

Bases: *UnconditionalGenerator*

MaliGANDiscriminator is LSTMs.

calculate_loss(*real_data*, *fake_data*)

Calculate the loss for real data and fake data. The discriminator is trained with the standard objective that GAN employs.

Parameters

- **real_data** (*torch.Tensor*) – The realistic sentence data, shape: [batch_size, max_seq_len].
- **fake_data** (*torch.Tensor*) – The generated sentence data, shape: [batch_size, max_seq_len].

Returns
The calculated loss of real data and fake data, shape: [].

Return type
torch.Tensor

forward(*data*)

Calculate the probability that the data is realistic.

Parameters
data (*torch.Tensor*) – The sentence data, shape: [batch_size, max_seq_len].

Returns
The probability that each sentence is realistic, shape: [batch_size].

Return type
torch.Tensor

training: **bool**

5.5.5 LeakGAN Discriminator

class textbox.module.Discriminator.LeakGANDiscriminator.**LeakGANDiscriminator**(*config*, *dataset*)

Bases: *UnconditionalGenerator*

CNN based discriminator for leakgan extracting feature of current sentence

calculate_loss(*real_data*, *fake_data*)

Calculate discriminator loss and acc

forward(*data*)

Get current sentence feature by CNN

get_feature(*inp*)

Get feature vector of given sentences

Parameters
inp – batch_size * max_seq_len

Returns
batch_size * feature_dim

highway(*data*)

training: **bool**

5.5.6 MaskGAN Discriminator

class textbox.module.Discriminator.MaskGANDiscriminator.**MaskGANDiscriminator**(*config*, *dataset*)

Bases: *GenerativeAdversarialNet*

RNN-based Encoder-Decoder architecture for MaskGAN discriminator

calculate_dis_loss(*fake_prediction*, *real_prediction*, *target_present*)

Compute Discriminator loss across real/fake

calculate_loss(*real_sequence*, *lengths*, *fake_sequence*, *targets_present*, *embedder*)

Calculate discriminator loss

create_critic_loss(*cumulative_rewards*, *estimated_values*, *target_present*)

Compute Critic loss in estimating the value function. This should be an estimate only for the missing elements.

critic(*fake_sequence*, *embedder*)

Define the Critic graph which is derived from the seq2seq Discriminator. This will be initialized with the same parameters as the language model and will share the forward RNN components with the Discriminator. This estimates the V(s_t), where the state s_t = x_0,. . . ,x_t-1.

Parameters
fake_sequence – sequence generated bs*seq_len

Returns
bs*seq_len

Return type
values

`forward`(*inputs*, *inputs_length*, *sequence*, *targets_present*, *embedder*)

Predict the real prob of the filled_in token using real sentence and fake sentence

Parameters

- **inputs** – real input bs*seq_len
- **inputs_length** – sentences length list[bs]
- **sequence** – real target or the generated sentence by Generator
- **targets_present** – target sentences present matrix bs*seq_len
- **embedder** – shared embedding with generator

Returns
the real prob of filled_in token predicted by discriminator

Return type
prediction

`mask_input`(*inputs*, *targets_present*)

Transforms the inputs to have missing tokens when it's masked out. The mask is for the targets, so therefore, to determine if an input at time t is masked, we have to check if the target at time t - 1 is masked out.

e.g.

- inputs = [a, b, c, d]
- targets = [b, c, d, e]
- targets_present = [1, 0, 1, 0]

then,

- masked_input = [a, b, <missing>, d]

Parameters

- **inputs** – Tensor of shape [batch_size, sequence_length]
- **targets_present** – Bool tensor of shape [batch_size, sequence_length] with 1 representing the presence of the word.

Returns

Tensor of shape [batch_size, sequence_length]
which takes on value of inputs when the input is present and takes on value=mask_token_idx to indicate a missing token.

Return type
masked_input

`mask_target_present`(*targets_present*, *lengths*)

`training: bool`

5.6 textbox.module.Embedder

5.6.1 Positional Embedding

class textbox.module.Embedder.position_embedder.**LearnedPositionalEmbedding**(*embedding_size*, *max_length=512*)

Bases: Module

This module produces Learned Positional Embedding.

forward(*input_seq*, *offset=0*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

training: bool

class textbox.module.Embedder.position_embedder.**SinusoidalPositionalEmbedding**(*embedding_size*, *max_length=512*)

Bases: Module

This module produces sinusoidal positional embeddings of any length.

forward(*input_seq*, *offset=0*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

static get_embedding(*max_length*, *embedding_size*)

Build sinusoidal embeddings. This matches the implementation in tensor2tensor, but differs slightly from the description in Section 3.5 of "Attention Is All You Need".

training: bool

5.7 textbox.module.Encoder

5.7.1 CNN Encoder

class textbox.module.Encoder.cnn_encoder.**BasicCNNEncoder**(*input_size*, *latent_size*)

Bases: Module

Basic Convolution Neural Network (CNN) encoder. Code reference: https://github.com/rohithreddy024/VAE-Text-Generation/

forward(*input*)

Implement the encoding process.

Parameters
input (*Torch.Tensor*) – source sequence embedding, shape: [batch_size, sequence_length, embedding_size].

Returns
output features, shape: [batch_size, sequence_length, feature_size].

Return type
torch.Tensor

training: **bool**

5.7.2 RNN Encoder

class textbox.module.Encoder.rnn_encoder.**BasicRNNEncoder**(*embedding_size*, *hidden_size*, *num_enc_layers*, *rnn_type*, *dropout_ratio*, *bidirectional=True*)

Bases: Module

Basic Recurrent Neural Network (RNN) encoder.

forward(*input_embeddings*, *input_length*, *hidden_states=None*)

Implement the encoding process.

Parameters

- **input_embeddings** (*Torch.Tensor*) – source sequence embedding, shape: [batch_size, sequence_length, embedding_size].
- **input_length** (*Torch.Tensor*) – length of input sequence, shape: [batch_size].
- **hidden_states** (*Torch.Tensor*) – initial hidden states, default: None.

Returns

- Torch.Tensor: output features, shape: [batch_size, sequence_length, num_directions * hidden_size].
- Torch.Tensor: hidden states, shape: [num_layers * num_directions, batch_size, hidden_size].

Return type
tuple

init_hidden(*input_embeddings*)

Initialize initial hidden states of RNN.

Parameters
input_embeddings (*Torch.Tensor*) – input sequence embedding, shape: [batch_size, sequence_length, embedding_size].

Returns
the initial hidden states.

Return type
Torch.Tensor

training: **bool**

5.7.3 Transformer Encoder

class textbox.module.Encoder.transformer_encoder.**TransformerEncoder**(*embedding_size*, *ffn_size*, *num_enc_layers*, *num_heads*, *attn_dropout_ratio=0.0*, *attn_weight_dropout_ratio=0.0*, *ffn_dropout_ratio=0.0*)

Bases: Module

The stacked Transformer encoder layers.

forward(*x*, *kv=None*, *self_padding_mask=None*, *output_all_encoded_layers=False*)

Implement the encoding process step by step.

Parameters

- **x** (*Torch.Tensor*) – target sequence embedding, shape: [batch_size, sequence_length, embedding_size].
- **kv** (*Torch.Tensor*) – the cached history latent vector, shape: [batch_size, sequence_length, embedding_size], default: None.
- **self_padding_mask** (*Torch.Tensor*) – padding mask of target sequence, shape: [batch_size, sequence_length], default: None.
- **output_all_encoded_layers** (*Bool*) – whether to output all the encoder layers, default: False.

Returns
output features, shape: [batch_size, sequence_length, ffn_size].

Return type
Torch.Tensor

training: bool

5.8 textbox.module.Generator

5.8.1 TextGAN Generator

class textbox.module.Generator.TextGANGenerator.**TextGANGenerator**(*config*, *dataset*)

Bases: *UnconditionalGenerator*

The generator of TextGAN.

adversarial_loss(*real_data*, *discriminator_func*)

Calculate the adversarial generator loss of real_data guided by discriminator_func.

Parameters

- **real_data** (*torch.Tensor*) – The realistic sentence data, shape: [batch_size, max_seq_len].
- **discriminator_func** (*function*) – The function provided from discriminator to calculated the loss of generated sentence.

Returns
The calculated adversarial loss, shape: [].

Return type
torch.Tensor

`calculate_loss`(*corpus*, *nll_test=False*)

Calculate the generated loss of corpus.

Parameters

- **corpus** (*Corpus*) – The corpus to be calculated.
- **nll_test** (*Bool*) – Optional; if nll_test is True the loss is calculated in sentence level rather than in word level.

Returns
The calculated loss of corpus, shape: [].

Return type
torch.Tensor

`generate`(*batch_data*, *eval_data*)

Generate tokens of sentences using eval_data.

Parameters

- **batch_data** (*Corpus*) – Single batch corpus information of evaluation data.
- **eval_data** – Common information of all evaluation data.

Returns
The generated tokens of each sentence.

Return type
List[List[str]]

`sample`()

Sample a batch of generated sentence indice.

Returns
The generated sentence indice, shape: [batch_size, max_length]. torch.Tensor: The latent code of the generated sentence, shape: [batch_size, hidden_size].

Return type
torch.Tensor

training: bool

5.8.2 SeqGAN Generator

class textbox.module.Generator.SeqGANGenerator.**SeqGANGenerator**(*config*, *dataset*)

Bases: *UnconditionalGenerator*

The generator of SeqGAN.

`adversarial_loss`(*discriminator_func*)

Calculate the adversarial generator loss guided by discriminator_func.

Parameters
discriminator_func (*function*) – The function provided from discriminator to calculated the loss of generated sentence.

Returns
The calculated adversarial loss, shape: [].

Return type
torch.Tensor

`calculate_loss`(*corpus*, *nll_test=False*)

Calculate the generated loss of corpus.

Parameters

- **corpus** (*Corpus*) – The corpus to be calculated.
- **nll_test** (*Bool*) – Optional; if nll_test is True the loss is calculated in sentence level rather than in word level.

Returns
The calculated loss of corpus, shape: [].

Return type
torch.Tensor

`generate`(*batch_data*, *eval_data*)

Generate tokens of sentences using eval_data.

Parameters

- **batch_data** (*Corpus*) – Single batch corpus information of evaluation data.
- **eval_data** – Common information of all evaluation data.

Returns
The generated tokens of each sentence.

Return type
List[List[str]]

`sample`(*sample_num*)

Sample sample_num generated sentence indice.

Parameters
sample_num (*int*) – The number to generate.

Returns
The generated sentence indice, shape: [sample_num, max_length].

Return type
torch.Tensor

`training: bool`

5.8.3 RankGAN Generator

`class textbox.module.Generator.RankGANGenerator.RankGANGenerator`(*config*, *dataset*)

Bases: *UnconditionalGenerator*

RankGANGenerator is a generative model with the LSTMs.

`adversarial_loss`(*ref_data*, *discriminator_func*)

Calculate the adversarial generator loss guided by discriminator. The Monte Carlo rollouts methods is utilized to simulate intermediate rewards when a sequence is incomplete. For the partial sequence, the average ranking score is used to approximate the expected future reward.

Parameters
discriminator_func (*function*) – The function provided from discriminator to calculated the ranking score.

Returns
The calculated adversarial loss, shape: [].

Return type
torch.Tensor

calculate_loss(*corpus*, *nll_test=False*)

Calculate the generated loss of corpus.

Parameters

- **corpus** (*Corpus*) – The corpus to be calculated.
- **nll_test** (*Bool*) – Optional; if nll_test is True the loss is calculated in sentence level rather than in word level.

Returns
The calculated loss of corpus, shape: [].

Return type
torch.Tensor

generate(*batch_data*, *eval_data*)

Generate tokens of sentences using eval_data.

Parameters

- **batch_data** (*Corpus*) – Single batch corpus information of evaluation data.
- **eval_data** – Common information of all evaluation data.

Returns
The generated tokens of each sentence.

Return type
List[List[str]]

sample(*sample_num*)

Sample sample_num generated sentence indice.

Parameters
sample_num (*int*) – The number to generate.

Returns
The generated sentence indice, shape: [sample_num, max_length].

Return type
torch.Tensor

sample_batch()

Sample a batch of generated sentence indice.

Returns
The generated sentence indice, shape: [batch_size, max_length].

Return type
torch.Tensor

training: bool

5.8.4 MaliGAN Generator

class textbox.module.Generator.MaliGANGenerator.**MaliGANGenerator**(*config*, *dataset*)

Bases: *UnconditionalGenerator*

MaliGANGenerator is a generative model with the LSTMs.

adversarial_loss(*discriminator_func*)

Calculate the adversarial generator loss guided by discriminator_func. A noval objective for the generator to optimize, using importance sampling. The training procedure is closer to maximum likelihood (MLE) training.

$$r_D(x) = \frac{D(x)}{1 - D(x)}$$

Parameters
discriminator_func (*function*) – The function provided from discriminator to calculated the loss of generated sentence.

Returns
The calculated adversarial loss, shape: [].

Return type
torch.Tensor

calculate_loss(*corpus*, *nll_test=False*)

Calculate the generated loss of corpus.

Parameters

- **corpus** (*Corpus*) – The corpus to be calculated.
- **nll_test** (*Bool*) – Optional; if nll_test is True the loss is calculated in sentence level rather than in word level.

Returns
The calculated loss of corpus, shape: [].

Return type
torch.Tensor

generate(*batch_data*, *eval_data*)

Generate tokens of sentences using eval_data.

Parameters

- **batch_data** (*Corpus*) – Single batch corpus information of evaluation data.
- **eval_data** – Common information of all evaluation data.

Returns
The generated tokens of each sentence.

Return type
List[List[str]]

sample(*sample_num*)

Sample sample_num generated sentence indice.

Parameters
sample_num (*int*) – The number to generate.

Returns
The generated sentence indice, shape: [sample_num, max_length].

Return type
torch.Tensor

`sample_batch()`

Sample a batch of generated sentence indice.

Returns
The generated sentence indice, shape: [batch_size, max_length].

Return type
torch.Tensor

`training: bool`

5.8.5 LeakGAN Generator

`class textbox.module.Generator.LeakGANGenerator.LeakGANGenerator(config, dataset)`

Bases: *UnconditionalGenerator*

LeakGAN generator consist of worker(LSTM) and manager(LSTM)

`adversarial_loss(dis)`

Generate data and calculate adversarial loss

`calculate_loss(targets, dis)`

Returns the nll test for predicting target sequence.

Parameters

- **targets** – target_idx(bs*seq_len) ,
- **dis** – discriminator model

Returns
the generator test nll

Return type
worker_loss

`forward(idx, inp, work_hidden, mana_hidden, feature, real_goal, train=False, pretrain=False)`

Embed input and sample on token at a time (seq_len = 1)

Parameters

- **idx** – index of current token in sentence
- **inp** – current input token for a batch [batch_size]
- **work_hidden** – 1 * batch_size * hidden_dim
- **mana_hidden** – 1 * batch_size * hidden_dim
- **feature** – 1 * batch_size * total_num_filters, feature of current sentence
- **real_goal** – batch_size * goal_out_size, real_goal in LeakGAN source code
- **train** – whether train or inference
- **pretrain** – whether pretrain or not pretrain

Returns
current output prob over vocab with log_softmax or softmax bs*vocab_size cur_goal: bs * 1 * goal_out_size work_hidden: 1 * batch_size * hidden_dim mana_hidden: 1 * batch_size * hidden_dim

Return type
out

`generate`(*batch_data*, *eval_data*, *dis*)

Generate sentences

`get_adv_loss`(*target*, *rewards*, *dis*)

Return a pseudo-loss that gives corresponding policy gradients (on calling .backward()). Inspired by the example in http://karpathy.github.io/2016/05/31/rl/

Args: target, rewards, dis, start_letter
target: batch_size * seq_len rewards: batch_size * seq_len (discriminator rewards for each token)

`get_reward_leakgan`(*sentences*, *rollout_num*, *dis*, *current_k=0*)

Get reward via Monte Carlo search for LeakGAN

Parameters

- **sentences** – size of batch_size * max_seq_len
- **rollout_num** – numbers of rollout
- **dis** – discriminator
- **current_k** – current training gen

Returns
batch_size * (max_seq_len / step_size)

Return type
reward

`init_hidden`(*batch_size=1*)

Init hidden state for lstm

`leakgan_forward`(*targets*, *dis*, *train=False*, *pretrain=False*)

Get all feature and goals according to given sentences

Parameters

- **targets** – batch_size * max_seq_len, pad eos token if the original sentence length less than max_seq_len
- **dis** – discriminator model
- **train** – if use temperature parameter
- **pretrain** – whether pretrain or not pretrain

Returns
batch_size * (seq_len + 1) * total_num_filter goal_array: batch_size * (seq_len + 1) * goal_out_size leak_out_array: batch_size * seq_len * vocab_size with log_softmax

Return type
feature_array

`leakgan_generate`(*targets*, *dis*, *train=False*)

manager_cos_loss(*batch_size*, *feature_array*, *goal_array*)

Get manager cosine distance loss

Returns
batch_size * (seq_len / step_size)

Return type
cos_loss

pretrain_loss(*corpus*, *dis*)

Return the generator pretrain loss for predicting target sequence.

Parameters

- **corpus** – target_text(bs*seq_len)
- **dis** – discriminator model

Returns
manager loss work_cn_loss: worker loss

Return type
manager_loss

redistribution(*idx*, *total*, *min_v*)

rescale(*reward*, *rollout_num=1.0*)

Rescale reward according to original paper

rollout_mc_search_leakgan(*targets*, *dis*, *given_num*)

Roll out to get mc search results

sample(*sample_num*, *dis*, *start_letter*, *train=False*)

Sample sentences

sample_batch()

Sample a batch of data

split_params()

Split parameter into worker and manager

training: bool

worker_cos_reward(*feature_array*, *goal_array*)

Get reward for worker (cosine distance)

Returns
batch_size * seq_len

Return type
cos_loss

worker_cross_entropy_loss(*target*, *leak_out_array*, *reduction='mean'*)

Get CrossEntropy loss for worker

worker_nll_loss(*target*, *leak_out_array*)

Get nll loss for worker

5.8.6 MaskGAN Generator

class textbox.module.Generator.MaskGANGenerator.**MaskGANGenerator**(*config*, *dataset*)

Bases: *GenerativeAdversarialNet*

RNN-based Encoder-Decoder architecture for maskgan generator

adversarial_loss(*inputs*, *lengths*, *targets*, *targets_present*, *discriminator*)

Calculate adversarial loss

calculate_loss(*logits*, *target_inputs*)

Calculate nll test loss

calculate_reinforce_objective(*log_probs*, *dis_predictions*, *mask_present*, *estimated_values=None*)

Calculate the REINFORCE objectives. The REINFORCE objective should only be on the tokens that were missing. Specifically, the final Generator reward should be based on the Discriminator predictions on missing tokens. The log probabilities should be only for missing tokens and the baseline should be calculated only on the missing tokens. For this model, we optimize the reward is the log of the *conditional* probability the Discriminator assigns to the distribution. Specifically, for a Discriminator D which outputs probability of real, given the past context, r_t = log D(x_t|x_0,x_1,... x_{t-1}) And the policy for Generator G is the log-probability of taking action x2 given the past context.

Parameters

- **log_probs** – Tensor of log probabilities of the tokens selected by the Generator. Shape [batch_size, sequence_length].
- **dis_predictions** – Tensor of the predictions from the Discriminator. Shape [batch_size, sequence_length].
- **present** – Tensor indicating which tokens are present. Shape [batch_size, sequence_length].
- **estimated_values** – Tensor of estimated state values of tokens. Shape [batch_size, sequence_length]

Returns

Final REINFORCE objective for the sequence. rewards: Tensor of rewards for sequence of shape [batch_size, sequence_length] advantages: Tensor of advantages for sequence of shape [batch_size, sequence_length] baselines: Tensor of baselines for sequence of shape [batch_size, sequence_length] maintain_averages_op: ExponentialMovingAverage apply average op to maintain the baseline.

Return type

final_gen_objective

calculate_train_loss(*inputs*, *lengths*, *targets*, *targets_present*, *validate=False*)

Calculate train loss for generator

create_critic_loss(*cumulative_rewards*, *estimated_values*, *target_present*)

Compute Critic loss in estimating the value function. This should be an estimate only for the missing elements.

forward(*inputs*, *input_length*, *targets*, *targets_present*, *pretrain=False*, *validate=False*)

Input real padded input and target sentence which not start from sos and end with eos(According to origin code). And input length used for LSTM

Parameters

- **inputs** – bs*seq_len

- **input_length** – list[bs]
- **targets_present** – target present matrix bs*seq_len 1: not mask 0: mask
- **pretrain** – control whether LM pretrain

Returns
samples log_probs: log prob logits: logits

Return type
output

generate(*batch_data*, *eval_data*)

Sample sentence

mask_cross_entropy_loss(*targets*, *logits*, *targets_present*)

Calculate the filling token cross entropy loss

mask_input(*inputs*, *targets_present*)

Transforms the inputs to have missing tokens when it's masked out. The mask is for the targets, so therefore, to determine if an input at time t is masked, we have to check if the target at time t - 1 is masked out.

e.g.

- inputs = [a, b, c, d]
- targets = [b, c, d, e]
- targets_present = [1, 0, 1, 0]

then,

- masked_input = [a, b, <missing>, d]

Parameters

- **inputs** – Tensor of shape [batch_size, sequence_length]
- **targets_present** – Bool tensor of shape [batch_size, sequence_length] with 1 representing the presence of the word.

Returns

Tensor of shape [batch_size, sequence_length]
which takes on value of inputs when the input is present and takes on value=mask_token_idx to indicate a missing token.

Return type
masked_input

training: bool

5.9 textbox.module.Optimizer

5.9.1 Optimizer

class textbox.module.Optimizer.optim.**AbstractOptim**(*base_optimizer: Optimizer*, *init_lr: float*)

Bases: object

load_state_dict(*state_dict: tuple*)

property lr

Get learning rate for current step.

state_dict()

step()

class textbox.module.Optimizer.optim.**ConstantOptim**(*base_optimizer: Optimizer*, *init_lr: float*, *max_lr: float*, *n_warmup_steps: int*)

Bases: *AbstractOptim*

property lr

Get learning rate for current step.

class textbox.module.Optimizer.optim.**CosineOptim**(*base_optimizer: Optimizer*, *init_lr: float*, *max_lr: float*, *n_warmup_steps: int*, *max_steps: int*)

Bases: *AbstractOptim*

property lr

Get learning rate for current step.

class textbox.module.Optimizer.optim.**InverseSquareRootOptim**(*base_optimizer: Optimizer*, *init_lr: float*, *max_lr: float*, *n_warmup_steps: int*)

Bases: *AbstractOptim*

property lr

Get learning rate for current step.

class textbox.module.Optimizer.optim.**LinearOptim**(*base_optimizer: Optimizer*, *init_lr: float*, *max_lr: float*, *n_warmup_steps: int*, *max_steps: int*)

Bases: *AbstractOptim*

property lr

Get learning rate for current step.

CHAPTER

SIX

TEXTBOX.QUICK_START

6.1 textbox.quick_start

textbox.quick_start.quick_start.**run_textbox**(*model=None*, *dataset=None*, *config_file_list=None*, *config_dict=None*, *saved=True*)

A fast running api, which includes the complete process of training and testing a model on a specified dataset

Parameters

- **model** (*str*) – model name
- **dataset** (*str*) – dataset name
- **config_file_list** (*list*) – config files used to modify experiment parameters
- **config_dict** (*dict*) – parameters dictionary used to modify experiment parameters
- **saved** (*bool*) – whether to save the model

CHAPTER

SEVEN

TEXTBOX.TRAINER

CHAPTER

EIGHT

TEXTBOX.UTILS

8.1 textbox.utils.enum_type

class textbox.utils.enum_type.**ModelType**(*value*)

Bases: Enum

Type of models.

- UNCONDITIONAL: Unconditional Generator
- GAN: Generative Adversarial Net
- SEQ2SEQ: Seq2Seq Generator
- ATTRIBUTE: Attribute Generator

ATTRIBUTE = 4

GAN = 2

SEQ2SEQ = 3

UNCONDITIONAL = 1

class textbox.utils.enum_type.**SpecialTokens**

Bases: object

Special tokens, including *PAD*, *UNK*, BOS, *EOS*. These tokens will by default have token ids 0, 1, 2, 3, respectively.

EOS = '<|endoftext|>'

PAD = '<|pad|>'

SOS = '<|startoftext|>'

UNK = '<|unk|>'

8.2 textbox.utils.logger

textbox.utils.logger.**init_logger**(*config*)

A logger that can show a message on standard output and write it into the file named *filename* simultaneously. All the message that you want to log MUST be str.

Parameters

config (Config) – An instance object of Config, used to record parameter information.

Example

```
>>> logger = logging.getLogger(config)
>>> logger.debug(train_state)
>>> logger.info(train_result)
```

8.3 textbox.utils.utils

textbox.utils.utils.**early_stopping**(*value*, *best*, *cur_step*, *max_step*, *bigger=True*)

validation-based early stopping

Parameters

- **value** (*float*) – current result
- **best** (*float*) – best result
- **cur_step** (*int*) – the number of consecutive steps that did not exceed the best result
- **max_step** (*int*) – threshold steps for stopping
- **bigger** (*bool, optional*) – whether the bigger the better

Returns

- float, best result after this step
- int, the number of consecutive steps that did not exceed the best result after this step
- bool, whether to stop
- bool, whether to update

Return type

tuple

textbox.utils.utils.**ensure_dir**(*dir_path*)

Make sure the directory exists, if it does not exist, create it

Parameters

dir_path (*str*) – directory path

textbox.utils.utils.**get_local_time**()

Get current time

Returns

current time

Return type
str

textbox.utils.utils.**get_model**(*model_name*)

Automatically select model class based on model name

Parameters
model_name (*str*) – model name

Returns
model class

Return type
Generator

textbox.utils.utils.**get_trainer**(*model_type*, *model_name*)

Automatically select trainer class based on model type and model name

Parameters

- **model_type** (ModelType) – model type
- **model_name** (*str*) – model name

Returns
trainer class

Return type
Trainer

textbox.utils.utils.**init_seed**(*seed*, *reproducibility*)

init random seed for random functions in numpy, torch, cuda and cudnn

Parameters

- **seed** (*int*) – random seed
- **reproducibility** (*bool*) – Whether to require reproducibility

CHAPTER

NINE

INDICES AND TABLES

- genindex
- search

PYTHON MODULE INDEX

t

INDEX

D

E

F

G

P

T

U

W

X