CRSLab

Release 0.1.1

RUC AIBox

Mar 28, 2021

API REFERENCE

CHAPTER

ONE

CRSLAB.QUICK_START PACKAGE

1.1 Submodules

1.2 Module contents

CHAPTER

TWO

CRSLAB.CONFIG PACKAGE

2.1 Submodules

class crslab.config.config.**Config**(*config_file*, *gpu='-1'*, *debug=False*)
Bases: object

Configurator module that load the defined parameters.

Load parameters and set log level.

Parameters

- **config_file** (*str*) – path to the config file, which should be in yaml format. You can use default config provided in the Github repo, or write it by yourself.
- **debug** (*bool, optional*) – whether to enable debug function during running. Defaults to False.

get(*item*, *default=None*)
Get value of corrsponding item in config

Parameters

- **item** (*str*) – key to query in config
- **default** (*optional*) – default value for item if not found in config. Defaults to None.

Returns value of corrsponding item in config

static load_yaml_configs(*filename*)
This function reads yaml file to build config dictionary

Parameters **filename** (*str*) – path to yaml config

Returns config

Return type dict

2.2 Module contents

Config module which loads parameters for the whole system.

crslab.config.**SAVE_PATH**
where system to save.

Type str

crslab.config.**DATASET_PATH**
where dataset to save.

Type str

crslab.config.**MODEL_PATH**
where model related data to save.

Type str

crslab.config.**PRETRAIN_PATH**
where pretrained model to save.

Type str

crslab.config.**EMBEDDING_PATH**
where pretrained embedding to save, used for evaluate embedding related metrics.

Type str

CHAPTER

THREE

CRSLAB.DATA PACKAGE

3.1 Subpackages

3.1.1 crslab.data.dataloader package

Submodules

class crslab.data.dataloader.base.**BaseDataLoader**(*opt*, *dataset*)
Bases: abc.ABC

Abstract class of dataloader

Notes

'scale' can be set in config to limit the size of dataset.

Parameters

- **opt** (*Config or dict*) – config for dataloader or the whole system.
- **dataset** – dataset

conv_batchify(*batch*)
batchify data for conversation after process.

Parameters **batch** (*list*) – processed batch dataset.

Returns batch data for the system to train conversation part.

conv_interact(*data*)
Process user input data for system to converse.

Parameters **data** – user input data.

Returns data for system in converse.

conv_process_fn()
Process whole data for conversation before batch_fn.

Returns processed dataset. Defaults to return the same as *self.dataset*.

get_conv_data(*batch_size*, *shuffle=True*)
get_data wrapper for conversation.

You can implement your own process_fn in conv_process_fn, batch_fn in conv_batchify.

Parameters

- **batch_size** (*int*) –
- **shuffle** (*bool, optional*) – Defaults to True.

Yields *tuple or dict of torch.Tensor* – batch data for conversation.

get_data(*batch_fn*, *batch_size*, *shuffle=True*, *process_fn=None*)
Collate batch data for system to fit

Parameters

- **batch_fn** (*func*) – function to collate data
- **batch_size** (*int*) –
- **shuffle** (*bool, optional*) – Defaults to True.
- **process_fn** (*func, optional*) – function to process dataset before batchify. Defaults to None.

Yields *tuple or dict of torch.Tensor* – batch data for system to fit

get_policy_data(*batch_size*, *shuffle=True*)
get_data wrapper for policy.

You can implement your own process_fn in `self.policy_process_fn`, batch_fn in `policy_batchify`.

Parameters

- **batch_size** (*int*) –
- **shuffle** (*bool, optional*) – Defaults to True.

Yields *tuple or dict of torch.Tensor* – batch data for policy.

get_rec_data(*batch_size*, *shuffle=True*)
get_data wrapper for recommendation.

You can implement your own process_fn in `rec_process_fn`, batch_fn in `rec_batchify`.

Parameters

- **batch_size** (*int*) –
- **shuffle** (*bool, optional*) – Defaults to True.

Yields *tuple or dict of torch.Tensor* – batch data for recommendation.

policy_batchify(*batch*)
batchify data for policy after process.

Parameters **batch** (*list*) – processed batch dataset.

Returns batch data for the system to train policy part.

policy_process_fn()
Process whole data for policy before batch_fn.

Returns processed dataset. Defaults to return the same as *self.dataset*.

rec_batchify(*batch*)
batchify data for recommendation after process.

Parameters **batch** (*list*) – processed batch dataset.

Returns batch data for the system to train recommendation part.

rec_interact(*data*)
process user input data for system to recommend.

Parameters **data** – user input data.

Returns data for system to recommend.

rec_process_fn()
Process whole data for recommendation before batch_fn.

Returns processed dataset. Defaults to return the same as *self.dataset*.

retain_recommender_target()
keep data whose role is recommender.

Returns Recommender part of `self.dataset`.

class crslab.data.dataloader.kbrd.**KBRDDataLoader**(*opt*, *dataset*, *vocab*)
Bases: *crslab.data.dataloader.base.BaseDataLoader*

Dataloader for model KBRD.

Notes

You can set the following parameters in config:

- `'context_truncate'`: the maximum length of context.
- `'response_truncate'`: the maximum length of response.
- `'entity_truncate'`: the maximum length of mentioned entities in context.

The following values must be specified in `vocab`:

- `'pad'`
- `'start'`
- `'end'`
- `'pad_entity'`

the above values specify the id of needed special token.

Parameters

- **opt** (*Config or dict*) – config for dataloader or the whole system.
- **dataset** – data for model.
- **vocab** (*dict*) – all kinds of useful size, idx and map between token and idx.

conv_batchify(*batch*)
batchify data for conversation after process.

Parameters **batch** (*list*) – processed batch dataset.

Returns batch data for the system to train conversation part.

conv_process_fn(**args*, ***kwargs*)
Process whole data for conversation before batch_fn.

Returns processed dataset. Defaults to return the same as *self.dataset*.

policy_batchify(**args*, ***kwargs*)
batchify data for policy after process.

Parameters **batch** (*list*) – processed batch dataset.

Returns batch data for the system to train policy part.

rec_batchify(*batch*)
batchify data for recommendation after process.

Parameters **batch** (*list*) – processed batch dataset.

Returns batch data for the system to train recommendation part.

rec_process_fn()
Process whole data for recommendation before batch_fn.

Returns processed dataset. Defaults to return the same as *self.dataset*.

class crslab.data.dataloader.kgsf.**KGSFDataLoader**(*opt*, *dataset*, *vocab*)
Bases: *crslab.data.dataloader.base.BaseDataLoader*

Dataloader for model KGSF.

Notes

You can set the following parameters in config:

- `'context_truncate'`: the maximum length of context.
- `'response_truncate'`: the maximum length of response.
- `'entity_truncate'`: the maximum length of mentioned entities in context.
- `'word_truncate'`: the maximum length of mentioned words in context.

The following values must be specified in `vocab`:

- `'pad'`
- `'start'`
- `'end'`
- `'pad_entity'`
- `'pad_word'`

the above values specify the id of needed special token.

- `'n_entity'`: the number of entities in the entity KG of dataset.

Parameters

- **opt** (*Config or dict*) – config for dataloader or the whole system.
- **dataset** – data for model.
- **vocab** (*dict*) – all kinds of useful size, idx and map between token and idx.

conv_batchify(*batch*)
batchify data for conversation after process.

Parameters **batch** (*list*) – processed batch dataset.

Returns batch data for the system to train conversation part.

conv_process_fn(**args*, ***kwargs*)
Process whole data for conversation before batch_fn.

Returns processed dataset. Defaults to return the same as *self.dataset*.

get_pretrain_data(*batch_size*, *shuffle=True*)

policy_batchify(**args*, ***kwargs*)
batchify data for policy after process.

Parameters **batch** (*list*) – processed batch dataset.

Returns batch data for the system to train policy part.

pretrain_batchify(*batch*)

rec_batchify(*batch*)
batchify data for recommendation after process.

Parameters **batch** (*list*) – processed batch dataset.

Returns batch data for the system to train recommendation part.

rec_process_fn()
Process whole data for recommendation before batch_fn.

Returns processed dataset. Defaults to return the same as *self.dataset*.

class crslab.data.dataloader.redial.**ReDialDataLoader**(*opt*, *dataset*, *vocab*)
Bases: *crslab.data.dataloader.base.BaseDataLoader*

Dataloader for model ReDial.

Notes

You can set the following parameters in config:

- `'utterance_truncate'`: the maximum length of a single utterance.
- `'conversation_truncate'`: the maximum length of the whole conversation.

The following values must be specified in `vocab`:

- `'pad'`
- `'start'`
- `'end'`
- `'unk'`

the above values specify the id of needed special token.

- `'ind2tok'`: map from index to token.
- `'n_entity'`: number of entities in the entity KG of dataset.
- `'vocab_size'`: size of vocab.

Parameters

- **opt** (*Config or dict*) – config for dataloader or the whole system.
- **dataset** – data for model.
- **vocab** (*dict*) – all kinds of useful size, idx and map between token and idx.

conv_batchify(*batch*)
batchify data for conversation after process.

Parameters **batch** (*list*) – processed batch dataset.

Returns batch data for the system to train conversation part.

conv_process_fn()
Process whole data for conversation before batch_fn.

Returns processed dataset. Defaults to return the same as *self.dataset*.

policy_batchify(*batch*)
batchify data for policy after process.

Parameters **batch** (*list*) – processed batch dataset.

Returns batch data for the system to train policy part.

rec_batchify(*batch*)
batchify data for recommendation after process.

Parameters **batch** (*list*) – processed batch dataset.

Returns batch data for the system to train recommendation part.

rec_process_fn(**args*, ***kwargs*)
Process whole data for recommendation before batch_fn.

Returns processed dataset. Defaults to return the same as *self.dataset*.

class crslab.data.dataloader.tgredial.**TGReDialDataLoader**(*opt*, *dataset*, *vocab*)
Bases: *crslab.data.dataloader.base.BaseDataLoader*

Dataloader for model TGReDial.

Notes

You can set the following parameters in config:

- `'context_truncate'`: the maximum length of context.
- `'response_truncate'`: the maximum length of response.
- `'entity_truncate'`: the maximum length of mentioned entities in context.
- `'word_truncate'`: the maximum length of mentioned words in context.
- `'item_truncate'`: the maximum length of mentioned items in context.

The following values must be specified in `vocab`:

- `'pad'`
- `'start'`
- `'end'`
- `'unk'`
- `'pad_entity'`
- `'pad_word'`

the above values specify the id of needed special token.

- `'ind2tok'`: map from index to token.

- `'tok2ind'`: map from token to index.
- `'vocab_size'`: size of vocab.
- `'id2entity'`: map from index to entity.
- `'n_entity'`: number of entities in the entity KG of dataset.
- `'sent_split'` (optional): token used to split sentence. Defaults to `'end'`.
- `'word_split'` (optional): token used to split word. Defaults to `'end'`.
- `'pad_topic'` (optional): token used to pad topic.
- `'ind2topic'` (optional): map from index to topic.

Parameters

- **opt** (*Config or dict*) – config for dataloader or the whole system.
- **dataset** – data for model.
- **vocab** (*dict*) – all kinds of useful size, idx and map between token and idx.

conv_batchify(*batch*)
batchify data for conversation after process.

Parameters **batch** (*list*) – processed batch dataset.

Returns batch data for the system to train conversation part.

conv_interact(*data*)
Process user input data for system to converse.

Parameters **data** – user input data.

Returns data for system in converse.

policy_batchify(*batch*)
batchify data for policy after process.

Parameters **batch** (*list*) – processed batch dataset.

Returns batch data for the system to train policy part.

policy_process_fn(**args*, ***kwargs*)
Process whole data for policy before batch_fn.

Returns processed dataset. Defaults to return the same as *self.dataset*.

rec_batchify(*batch*)
batchify data for recommendation after process.

Parameters **batch** (*list*) – processed batch dataset.

Returns batch data for the system to train recommendation part.

rec_interact(*data*)
process user input data for system to recommend.

Parameters **data** – user input data.

Returns data for system to recommend.

rec_process_fn(**args*, ***kwargs*)
Process whole data for recommendation before batch_fn.

Returns processed dataset. Defaults to return the same as *self.dataset*.

crslab.data.dataloader.utils.**add_start_end_token_idx**(*vec: list*, *start_token_idx: Optional[int] = None*, *end_token_idx: Optional[int] = None*)

Can choose to add start token in the beginning and end token in the end.

Parameters

- **vec** – source list composed of indexes.
- **start_token_idx** – index of start token.
- **end_token_idx** – index of end token.

Returns list added start or end token index.

Return type list

crslab.data.dataloader.utils.**get_onehot**(*data_list*, *categories*) → torch.Tensor

Transform lists of label into one-hot.

Parameters

- **data_list** (*list of list of int*) – source data.
- **categories** (*int*) – #label class.

Returns one-hot labels.

Return type torch.Tensor

crslab.data.dataloader.utils.**merge_utt**(*conversation*, *split_token_idx=None*, *keep_split_in_tail=False*, *final_token_idx=None*)

merge utterances in one conversation.

Parameters

- **conversation** (*list of list of int*) – conversation consist of utterances consist of tokens.
- **split_token_idx** (*int*) – index of split token. Defaults to None.
- **keep_split_in_tail** (*bool*) – split in tail or head. Defaults to False.
- **final_token_idx** (*int*) – index of final token. Defaults to None.

Returns tokens of all utterances in one list.

Return type list

crslab.data.dataloader.utils.**padded_tensor**(*items: List[Union[List[int], torch.LongTensor]]*, *pad_idx: int = 0*, *pad_tail: bool = True*, *max_len: Optional[int] = None*) → torch.LongTensor

Create a padded matrix from an uneven list of lists.

Returns padded matrix.

Matrix is right-padded (filled to the right) by default, but can be left padded if the flag is set to True.

Matrix can also be placed on cuda automatically.

Parameters

- **items** (*list[iter[int]]*) – List of items

- **pad_idx** (*int*) – the value to use for padding
- **pad_tail** (*bool*) –
- **max_len** (*int*) – if None, the max length is the maximum item length

Returns padded tensor.

Return type Tensor[int64]

crslab.data.dataloader.utils.**truncate**(*vec*, *max_length*, *truncate_tail=True*)

truncate vec to make its length no more than max length.

Parameters

- **vec** (*list*) – source list.
- **max_length** (*int*) –
- **truncate_tail** (*bool, optional*) – Defaults to True.

Returns truncated vec.

Return type list

Module contents

3.1.2 crslab.data.dataset package

Subpackages

crslab.data.dataset.durecdial package

Submodules

DuRecDial

References

Liu, Zeming, et al. "Towards Conversational Recommendation over Multi-Type Dialogs." in ACL 2020.

class crslab.data.dataset.durecdial.durecdial.**DuRecDialDataset**(*opt*, *tokenize*, *restore=False*, *save=False*)

Bases: crslab.data.dataset.base.BaseDataset

train_data
train dataset.

valid_data
valid dataset.

test_data
test dataset.

vocab

```
{
    'tok2ind': map from token to index,
    'ind2tok': map from index to token,
    'entity2id': map from entity to index,
    'id2entity': map from index to entity,
    'word2id': map from word to index,
    'vocab_size': len(self.tok2ind),
    'n_entity': max(self.entity2id.values()) + 1,
    'n_word': max(self.word2id.values()) + 1,
}
```

Type dict

Notes

`'unk'` must be specified in `'special_token_idx'` in `resources.py`.

Parameters

- **opt** (*Config or dict*) – config for dataset or the whole system.
- **tokenize** (*str*) – how to tokenize dataset.
- **restore** (*bool*) – whether to restore saved dataset which has been processed. Defaults to False.
- **save** (*bool*) – whether to save dataset after processing. Defaults to False.

Module contents

crslab.data.dataset.gorecdial package

Submodules

GoRecDial

References

Kang, Dongyeop, et al. "Recommendation as a Communication Game: Self-Supervised Bot-Play for Goal-oriented Dialogue." in EMNLP 2019.

class crslab.data.dataset.gorecdial.gorecdial.**GoRecDialDataset**(*opt*, *tokenize*, *restore=False*, *save=False*)

Bases: crslab.data.dataset.base.BaseDataset

train_data
train dataset.

valid_data
valid dataset.

test_data
test dataset.

vocab

```
{
    'tok2ind': map from token to index,
    'ind2tok': map from index to token,
    'entity2id': map from entity to index,
    'id2entity': map from index to entity,
    'word2id': map from word to index,
    'vocab_size': len(self.tok2ind),
    'n_entity': max(self.entity2id.values()) + 1,
    'n_word': max(self.word2id.values()) + 1,
}
```

Type dict

Notes

`'unk'` must be specified in `'special_token_idx'` in `resources.py`.

Specify tokenized resource and init base dataset.

Parameters

- **opt** (*Config or dict*) – config for dataset or the whole system.
- **tokenize** (*str*) – how to tokenize dataset.
- **restore** (*bool*) – whether to restore saved dataset which has been processed. Defaults to False.
- **save** (*bool*) – whether to save dataset after processing. Defaults to False.

Module contents

crslab.data.dataset.inspired package

Submodules

Inspired

References

Hayati, Shirley Anugrah, et al. "INSPIRED: Toward Sociable Recommendation Dialog Systems." in EMNLP 2020.

class crslab.data.dataset.inspired.inspired.**InspiredDataset**(*opt*, *tokenize*, *restore=False*, *save=False*)

Bases: `crslab.data.dataset.base.BaseDataset`

train_data
train dataset.

valid_data
valid dataset.

test_data
test dataset.

vocab

```
{
    'tok2ind': map from token to index,
    'ind2tok': map from index to token,
    'entity2id': map from entity to index,
    'id2entity': map from index to entity,
    'word2id': map from word to index,
    'vocab_size': len(self.tok2ind),
    'n_entity': max(self.entity2id.values()) + 1,
    'n_word': max(self.word2id.values()) + 1,
}
```

Type dict

Notes

`'unk'` must be specified in `'special_token_idx'` in `resources.py`.

Specify tokenized resource and init base dataset.

Parameters

- **opt** (*Config or dict*) – config for dataset or the whole system.
- **tokenize** (*str*) – how to tokenize dataset.
- **restore** (*bool*) – whether to restore saved dataset which has been processed. Defaults to False.
- **save** (*bool*) – whether to save dataset after processing. Defaults to False.

Module contents

crslab.data.dataset.opendialkg package

Submodules

OpenDialKG

References

Moon, Seungwhan, et al. "Opendialkg: Explainable conversational reasoning with attention-based walks over knowledge graphs." in ACL 2019.

class crslab.data.dataset.opendialkg.opendialkg.**OpenDialKGDataset**(*opt*, *tokenize*, *restore=False*, *save=False*)

Bases: crslab.data.dataset.base.BaseDataset

train_data
train dataset.

valid_data
valid dataset.

test_data
test dataset.

vocab

```
{
    'tok2ind': map from token to index,
    'ind2tok': map from index to token,
    'entity2id': map from entity to index,
    'id2entity': map from index to entity,
    'word2id': map from word to index,
    'vocab_size': len(self.tok2ind),
    'n_entity': max(self.entity2id.values()) + 1,
    'n_word': max(self.word2id.values()) + 1,
}
```

Type dict

Notes

'unk' must be specified in 'special_token_idx' in resources.py.

Specify tokenized resource and init base dataset.

Parameters

- **opt** (*Config or dict*) – config for dataset or the whole system.
- **tokenize** (*str*) – how to tokenize dataset.
- **restore** (*bool*) – whether to restore saved dataset which has been processed. Defaults to False.
- **save** (*bool*) – whether to save dataset after processing. Defaults to False.

Module contents

crslab.data.dataset.redial package

Submodules

ReDial

References

Li, Raymond, et al. "Towards deep conversational recommendations." in NeurIPS 2018.

class crslab.data.dataset.redial.redial.**ReDialDataset**(*opt*, *tokenize*, *restore=False*, *save=False*)

Bases: crslab.data.dataset.base.BaseDataset

train_data
train dataset.

valid_data
valid dataset.

test_data
test dataset.

vocab

```
{
    'tok2ind': map from token to index,
    'ind2tok': map from index to token,
    'entity2id': map from entity to index,
    'id2entity': map from index to entity,
    'word2id': map from word to index,
    'vocab_size': len(self.tok2ind),
    'n_entity': max(self.entity2id.values()) + 1,
    'n_word': max(self.word2id.values()) + 1,
}
```

Type dict

Notes

'unk' must be specified in 'special_token_idx' in resources.py.

Specify tokenized resource and init base dataset.

Parameters

- **opt** (*Config or dict*) – config for dataset or the whole system.
- **tokenize** (*str*) – how to tokenize dataset.
- **restore** (*bool*) – whether to restore saved dataset which has been processed. Defaults to False.

- **save** (*bool*) – whether to save dataset after processing. Defaults to False.

Module contents

crslab.data.dataset.tgredial package

Submodules

TGReDial

References

Zhou, Kun, et al. "Towards Topic-Guided Conversational Recommender System." in COLING 2020.

class crslab.data.dataset.tgredial.tgredial.**TGReDialDataset**(*opt*, *tokenize*, *restore=False*, *save=False*)

Bases: crslab.data.dataset.base.BaseDataset

train_data
train dataset.

valid_data
valid dataset.

test_data
test dataset.

vocab

```
{
    'tok2ind': map from token to index,
    'ind2tok': map from index to token,
    'topic2ind': map from topic to index,
    'ind2topic': map from index to topic,
    'entity2id': map from entity to index,
    'id2entity': map from index to entity,
    'word2id': map from word to index,
    'vocab_size': len(self.tok2ind),
    'n_topic': len(self.topic2ind) + 1,
    'n_entity': max(self.entity2id.values()) + 1,
    'n_word': max(self.word2id.values()) + 1,
}
```

Type dict

Notes

'unk' and 'pad_topic' must be specified in 'special_token_idx' in resources.py.

Specify tokenized resource and init base dataset.

Parameters

- **opt** (*Config or dict*) – config for dataset or the whole system.
- **tokenize** (*str*) – how to tokenize dataset.
- **restore** (*bool*) – whether to restore saved dataset which has been processed. Defaults to False.
- **save** (*bool*) – whether to save dataset after processing. Defaults to False.

Module contents

Submodules

Module contents

3.2 Module contents

Data module which reads, processes and batches data for the whole system

crslab.data.**dataset_register_table**
record all supported dataset

Type dict

crslab.data.**dataset_language_map**
record all dataset corresponding language

Type dict

crslab.data.**dataloader_register_table**
record all model corresponding dataloader

Type dict

crslab.data.**get_dataloader**(*opt*, *dataset*, *vocab*) → *crslab.data.dataloader.base.BaseDataLoader*
get dataloader to batchify dataset

Parameters

- **opt** (*Config or dict*) – config for dataloader or the whole system.
- **dataset** – processed raw data, no side data.
- **vocab** (*dict*) – all kinds of useful size, idx and map between token and idx.

Returns dataloader

crslab.data.**get_dataset**(*opt*, *tokenize*, *restore*, *save*) → crslab.data.dataset.base.BaseDataset
get and process dataset

Parameters

- **opt** (*Config or dict*) – config for dataset or the whole system.
- **tokenize** (*str*) – how to tokenize the dataset.
- **restore** (*bool*) – whether to restore saved dataset which has been processed.
- **save** (*bool*) – whether to save dataset after processing.

Returns processed dataset

CHAPTER

FOUR

CRSLAB.EVALUATOR PACKAGE

4.1 Subpackages

4.1.1 crslab.evaluator.metrics package

Submodules

Module contents

4.2 Submodules

4.3 Module contents

CHAPTER

FIVE

CRSLAB.MODEL PACKAGE

5.1 Subpackages

5.1.1 crslab.model.conversation package

Subpackages

crslab.model.conversation.gpt2 package

Submodules

GPT2

References

Radford, Alec, et al. "Language Models are Unsupervised Multitask Learners.".

class crslab.model.conversation.gpt2.gpt2.**GPT2Model**(*opt*, *device*, *vocab*, *side_data*)

Bases: *crslab.model.base.BaseModel*

context_truncate
A integer indicating the length of dialogue context.

response_truncate
A integer indicating the length of dialogue response.

pad_id
A integer indicating the id of padding token.

Parameters

- **opt** (*dict*) – A dictionary record the hyper parameters.
- **device** (*torch.device*) – A variable indicating which device to place the data and model.
- **vocab** (*dict*) – A dictionary record the vocabulary information.
- **side_data** (*dict*) – A dictionary record the side data.

build_model()
build model

calculate_loss(*logit*, *labels*)

Parameters

- **preds** – torch.FloatTensor, shape=(bs, response_truncate, vocab_size)
- **labels** – torch.LongTensor, shape=(bs, response_truncate)

forward(*batch*, *mode*)

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(*context*)

Parameters **context** – torch.tensor, shape=(bs, context_turncate)

Returns torch.tensor, shape=(bs, reponse_turncate-1)

Return type generated_response

generate_bs(*context*, *beam=4*)

Module contents

crslab.model.conversation.transformer package

Submodules

Transformer

References

Zhou, Kun, et al. "Towards Topic-Guided Conversational Recommender System." in COLING 2020.

---

class crslab.model.conversation.transformer.transformer.**TransformerModel**(*opt*, *device*, *vocab*, *side_data*)

Bases: *crslab.model.base.BaseModel*

vocab_size
A integer indicating the vocabulary size.

pad_token_idx
A integer indicating the id of padding token.

start_token_idx
A integer indicating the id of start token.

end_token_idx
A integer indicating the id of end token.

token_emb_dim
A integer indicating the dimension of token embedding layer.

pretrain_embedding
A string indicating the path of pretrained embedding.

n_word
A integer indicating the number of words.

n_entity
A integer indicating the number of entities.

pad_word_idx
A integer indicating the id of word padding.

pad_entity_idx
A integer indicating the id of entity padding.

num_bases
A integer indicating the number of bases.

kg_emb_dim
A integer indicating the dimension of kg embedding.

n_heads
A integer indicating the number of heads.

n_layers
A integer indicating the number of layer.

ffn_size
A integer indicating the size of ffn hidden.

dropout
A float indicating the drouput rate.

attention_dropout
A integer indicating the drouput rate of attention layer.

relu_dropout
A integer indicating the drouput rate of relu layer.

learn_positional_embeddings
A boolean indicating if we learn the positional embedding.

embeddings_scale
A boolean indicating if we use the embeddings scale.

reduction
A boolean indicating if we use the reduction.

n_positions
A integer indicating the number of position.

longest_label
A integer indicating the longest length for response generation.

Parameters

- **opt** (*dict*) – A dictionary record the hyper parameters.

- **device** (*torch.device*) – A variable indicating which device to place the data and model.
- **vocab** (*dict*) – A dictionary record the vocabulary information.
- **side_data** (*dict*) – A dictionary record the side data.

_starts(*batch_size*)
Return bsz start tokens.

build_model()
build model

forward(*batch*, *mode*)
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.

Module contents

Module contents

5.1.2 crslab.model.crs package

Subpackages

crslab.model.crs.kbrd package

Submodules

KBRD

References

Chen, Qibin, et al. "Towards Knowledge-Based Recommender Dialog System." in EMNLP 2019.

---

class crslab.model.crs.kbrd.kbrd.**KBRDModel**(*opt*, *device*, *vocab*, *side_data*)
Bases: *crslab.model.base.BaseModel*

vocab_size
A integer indicating the vocabulary size.

pad_token_idx
A integer indicating the id of padding token.

start_token_idx
A integer indicating the id of start token.

end_token_idx
A integer indicating the id of end token.

token_emb_dim
A integer indicating the dimension of token embedding layer.

pretrain_embedding
A string indicating the path of pretrained embedding.

n_entity
A integer indicating the number of entities.

n_relation
A integer indicating the number of relation in KG.

num_bases
A integer indicating the number of bases.

kg_emb_dim
A integer indicating the dimension of kg embedding.

user_emb_dim
A integer indicating the dimension of user embedding.

n_heads
A integer indicating the number of heads.

n_layers
A integer indicating the number of layer.

ffn_size
A integer indicating the size of ffn hidden.

dropout
A float indicating the dropout rate.

attention_dropout
A integer indicating the dropout rate of attention layer.

relu_dropout
A integer indicating the dropout rate of relu layer.

learn_positional_embeddings
A boolean indicating if we learn the positional embedding.

embeddings_scale
A boolean indicating if we use the embeddings scale.

reduction
A boolean indicating if we use the reduction.

n_positions
A integer indicating the number of position.

longest_label
A integer indicating the longest length for response generation.

user_proj_dim
A integer indicating dim to project for user embedding.

Parameters

- **opt** (*dict*) – A dictionary record the hyper parameters.

- **device** (*torch.device*) – A variable indicating which device to place the data and model.
- **vocab** (*dict*) – A dictionary record the vocabulary information.
- **side_data** (*dict*) – A dictionary record the side data.

_starts(*batch_size*)
Return bsz start tokens.

build_model(**args*, ***kwargs*)
build model

converse(*batch*, *mode*)
calculate loss and prediction of conversation for batch under certain mode

Parameters

- **batch** (*dict or tuple*) – batch data
- **mode** (*str, optional*) – train/valid/test.

decode_beam_search(*encoder_states*, *user_embedding*, *beam=4*)

decode_forced(*encoder_states*, *user_embedding*, *resp*)

decode_greedy(*encoder_states*, *user_embedding*)

encode_user(*entity_lists*, *kg_embedding*)

forward(*batch*, *mode*, *stage*)
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.

---

recommend(*batch*, *mode*)
calculate loss and prediction of recommendation for batch under certain mode

Parameters

- **batch** (*dict or tuple*) – batch data
- **mode** (*str, optional*) – train/valid/test.

Module contents

crslab.model.crs.kgsf package

Submodules

KGSF

References

Zhou, Kun, et al. "Improving Conversational Recommender Systems via Knowledge Graph based Semantic Fusion." in KDD 2020.

---

class crslab.model.crs.kgsf.kgsf.**KGSFModel**(*opt*, *device*, *vocab*, *side_data*)

Bases: *crslab.model.base.BaseModel*

vocab_size
A integer indicating the vocabulary size.

pad_token_idx
A integer indicating the id of padding token.

start_token_idx
A integer indicating the id of start token.

end_token_idx
A integer indicating the id of end token.

token_emb_dim
A integer indicating the dimension of token embedding layer.

pretrain_embedding
A string indicating the path of pretrained embedding.

n_word
A integer indicating the number of words.

n_entity
A integer indicating the number of entities.

pad_word_idx
A integer indicating the id of word padding.

pad_entity_idx
A integer indicating the id of entity padding.

num_bases
A integer indicating the number of bases.

kg_emb_dim
A integer indicating the dimension of kg embedding.

n_heads
A integer indicating the number of heads.

n_layers
A integer indicating the number of layer.

ffn_size
A integer indicating the size of ffn hidden.

dropout
A float indicating the dropout rate.

attention_dropout
A integer indicating the dropout rate of attention layer.

relu_dropout
A integer indicating the dropout rate of relu layer.

learn_positional_embeddings
A boolean indicating if we learn the positional embedding.

embeddings_scale
A boolean indicating if we use the embeddings scale.

reduction
A boolean indicating if we use the reduction.

n_positions
A integer indicating the number of position.

response_truncate = A integer indicating the longest length for response generation.

pretrained_embedding
A string indicating the path of pretrained embedding.

Parameters

- **opt** (*dict*) – A dictionary record the hyper parameters.
- **device** (*torch.device*) – A variable indicating which device to place the data and model.
- **vocab** (*dict*) – A dictionary record the vocabulary information.
- **side_data** (*dict*) – A dictionary record the side data.

_starts(*batch_size*)
Return bsz start tokens.

build_model()
build model

converse(*batch*, *mode*)
calculate loss and prediction of conversation for batch under certain mode

Parameters

- **batch** (*dict or tuple*) – batch data
- **mode** (*str, optional*) – train/valid/test.

forward(*batch*, *stage*, *mode*)
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.

---

freeze_parameters()

pretrain_infomax(*batch*)
words: (batch_size, word_length) entity_labels: (batch_size, n_entity)

recommend(*batch*, *mode*)
context_entities: (batch_size, entity_length) context_words: (batch_size, word_length) movie: (batch_size)

class crslab.model.crs.kgsf.modules.**GateLayer**(*input_dim*)
Bases: torch.nn.modules.module.Module

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*input1*, *input2*)
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.

---

class crslab.model.crs.kgsf.modules.**TransformerDecoderKG**(*n_heads*, *n_layers*, *embedding_size*, *ffn_size*, *vocabulary_size*, *embedding*, *dropout=0.0*, *attention_dropout=0.0*, *relu_dropout=0.0*, *embeddings_scale=True*, *learn_positional_embeddings=False*, *padding_idx=None*, *n_positions=1024*)

Bases: `torch.nn.modules.module.Module`

Transformer Decoder layer.

Parameters

- **n_heads** (*int*) – the number of multihead attention heads.
- **n_layers** (*int*) – number of transformer layers.
- **embedding_size** (*int*) – the embedding sizes. Must be a multiple of n_heads.
- **ffn_size** (*int*) – the size of the hidden layer in the FFN
- **embedding** – an embedding matrix for the bottom layer of the transformer. If none, one is created for this encoder.
- **dropout** (*float*) – Dropout used around embeddings and before layer layer normalizations. This is used in Vaswani 2017 and works well on large datasets.
- **attention_dropout** (*float*) – Dropout performed after the multhead attention softmax. This is not used in Vaswani 2017.
- **relu_dropout** (*float*) – Dropout used after the ReLU in the FFN. Not used in Vaswani 2017, but used in Tensor2Tensor.
- **padding_idx** (*int*) – Reserved padding index in the embeddings matrix.
- **learn_positional_embeddings** (*bool*) – If off, sinusoidal embeddings are used. If on, position embeddings are learned from scratch.
- **embeddings_scale** (*bool*) – Scale embeddings relative to their dimensionality. Found useful in fairseq.
- **n_positions** (*int*) – Size of the position embeddings matrix.

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*input*, *encoder_state*, *kg_encoder_output*, *kg_encoder_mask*, *db_encoder_output*, *db_encoder_mask*, *incr_state=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.

class crslab.model.crs.kgsf.modules.**TransformerDecoderLayerKG**(*n_heads*, *embedding_size*, *ffn_size*, *attention_dropout=0.0*, *relu_dropout=0.0*, *dropout=0.0*)

Bases: `torch.nn.modules.module.Module`

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*x*, *encoder_output*, *encoder_mask*, *kg_encoder_output*, *kg_encoder_mask*, *db_encoder_output*, *db_encoder_mask*)

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.

Module contents

crslab.model.crs.redial package

Submodules

class crslab.model.crs.redial.modules.**HRNN**(*utterance_encoder_hidden_size*, *dialog_encoder_hidden_size*, *dialog_encoder_num_layers*, *pad_token_idx*, *embedding=None*, *use_dropout=False*, *dropout=0.3*)

Bases: `torch.nn.modules.module.Module`

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*context*, *utterance_lengths*, *dialog_lengths*)

Parameters

- **context** – (batch_size, max_context_length, max_utterance_length)
- **utterance_lengths** – (batch_size, max_context_length)
- **dialog_lengths** – (batch_size)

Return context_state (batch_size, context_encoder_hidden_size)

get_utterance_encoding(*context*, *utterance_lengths*)

Parameters

- **context** – (batch_size, max_conversation_length, max_utterance_length)

- **utterance_lengths** – (batch_size, max_conversation_length)

Return utterance_encoding (batch_size, max_conversation_length, 2 * utterance_encoder_hidden_size)

class crslab.model.crs.redial.modules.**SwitchingDecoder**(*hidden_size*, *context_size*, *num_layers*, *vocab_size*, *embedding*, *pad_token_idx*)

Bases: torch.nn.modules.module.Module

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*request*, *request_lengths*, *context_state*)

Parameters

- **request** – (batch_size, max_utterance_length)
- **request_lengths** – (batch_size)
- **context_state** – (batch_size, context_encoder_hidden_size)

Return log_probabilities (batch_size, max_utterance_length, vocab_size + 1)

ReDial_Conv

References

Li, Raymond, et al. "Towards deep conversational recommendations." in NeurIPS.

class crslab.model.crs.redial.redial_conv.**ReDialConvModel**(*opt*, *device*, *vocab*, *side_data*)

Bases: *crslab.model.base.BaseModel*

vocab_size
A integer indicating the vocabulary size.

pad_token_idx
A integer indicating the id of padding token.

start_token_idx
A integer indicating the id of start token.

end_token_idx
A integer indicating the id of end token.

unk_token_idx
A integer indicating the id of unk token.

pretrained_embedding
A string indicating the path of pretrained embedding.

embedding_dim
A integer indicating the dimension of item embedding.

utterance_encoder_hidden_size
A integer indicating the size of hidden size in utterance encoder.

dialog_encoder_hidden_size
A integer indicating the size of hidden size in dialog encoder.

dialog_encoder_num_layers
A integer indicating the number of layers in dialog encoder.

use_dropout
A boolean indicating if we use the dropout.

dropout
A float indicating the dropout rate.

decoder_hidden_size
A integer indicating the size of hidden size in decoder.

decoder_num_layers
A integer indicating the number of layer in decoder.

decoder_embedding_dim
A integer indicating the dimension of embedding in decoder.

Parameters

- **opt** (*dict*) – A dictionary record the hyper parameters.
- **device** (*torch.device*) – A variable indicating which device to place the data and model.
- **vocab** (*dict*) – A dictionary record the vocabulary information.
- **side_data** (*dict*) – A dictionary record the side data.

build_model()
build model

forward(*batch*, *mode*)

Parameters **batch** –

```
{
    'context': (batch_size, max_context_length, max_utterance_
↪length),
    'context_lengths': (batch_size),
    'utterance_lengths': (batch_size, max_context_length),
    'request': (batch_size, max_utterance_length),
    'request_lengths': (batch_size),
    'response': (batch_size, max_utterance_length)
}
```

ReDial_Rec

References

Li, Raymond, et al. "Towards deep conversational recommendations." in NeurIPS.

class crslab.model.crs.redial.redial_rec.**ReDialRecModel**(*opt*, *device*, *vocab*, *side_data*)

Bases: *crslab.model.base.BaseModel*

n_entity
A integer indicating the number of entities.

layer_sizes
A integer indicating the size of layer in autorec.

pad_entity_idx
A integer indicating the id of entity padding.

Parameters

- **opt** (*dict*) – A dictionary record the hyper parameters.
- **device** (*torch.device*) – A variable indicating which device to place the data and model.
- **vocab** (*dict*) – A dictionary record the vocabulary information.
- **side_data** (*dict*) – A dictionary record the side data.

build_model()
build model

forward(*batch*, *mode*)

Parameters

- **batch** –

```
{
    'context_entities': (batch_size, n_entity),
    'item': (batch_size)
}
```

- **mode** (*str*) –

Module contents

crslab.model.crs.tgredial package

Submodules

TGReDial_Conv

References

Zhou, Kun, et al. "Towards Topic-Guided Conversational Recommender System." in COLING 2020.

class crslab.model.crs.tgredial.tg_conv.**TGConvModel**(*opt*, *device*, *vocab*, *side_data*)
Bases: *crslab.model.base.BaseModel*

context_truncate
A integer indicating the length of dialogue context.

response_truncate
A integer indicating the length of dialogue response.

pad_id
A integer indicating the id of padding token.

Parameters

- **opt** (*dict*) – A dictionary record the hyper parameters.
- **device** (*torch.device*) – A variable indicating which device to place the data and model.
- **vocab** (*dict*) – A dictionary record the vocabulary information.
- **side_data** (*dict*) – A dictionary record the side data.

build_model()
build model

calculate_loss(*logit*, *labels*)

Parameters

- **preds** – torch.FloatTensor, shape=(bs, response_truncate, vocab_size)
- **labels** – torch.LongTensor, shape=(bs, response_truncate)

forward(*batch*, *mode*)
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(*context*)

Parameters **context** – torch.tensor, shape=(bs, context_turncate)

Returns torch.tensor, shape=(bs, reponse_turncate-1)

Return type generated_response

generate_bs(*context*, *beam=4*)

TGReDial_Policy

References

Zhou, Kun, et al. "Towards Topic-Guided Conversational Recommender System." in COLING 2020.

class crslab.model.crs.tgredial.tg_policy.**TGPolicyModel**(*opt*, *device*, *vocab*, *side_data*)

Bases: *crslab.model.base.BaseModel*

Parameters

- **opt** (*dict*) – A dictionary record the hyper parameters.
- **device** (*torch.device*) – A variable indicating which device to place the data and model.
- **vocab** (*dict*) – A dictionary record the vocabulary information.
- **side_data** (*dict*) – A dictionary record the side data.

build_model(**args*, ***kwargs*)
build model

forward(*batch*, *mode*)
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.

TGReDial_Rec

References

Zhou, Kun, et al. "Towards Topic-Guided Conversational Recommender System." in COLING 2020.

---

class crslab.model.crs.tgredial.tg_rec.**TGRecModel**(*opt*, *device*, *vocab*, *side_data*)
Bases: *crslab.model.base.BaseModel*

hidden_dropout_prob
A float indicating the dropout rate to dropout hidden state in SASRec.

initializer_range
A float indicating the range of parameters initization in SASRec.

hidden_size
A integer indicating the size of hidden state in SASRec.

max_seq_length
A integer indicating the max interaction history length.

item_size
A integer indicating the number of items.

num_attention_heads
A integer indicating the head number in SASRec.

attention_probs_dropout_prob
A float indicating the dropout rate in attention layers.

hidden_act
A string indicating the activation function type in SASRec.

num_hidden_layers
A integer indicating the number of hidden layers in SASRec.

Parameters

- **opt** (*dict*) – A dictionary record the hyper parameters.
- **device** (*torch.device*) – A variable indicating which device to place the data and model.
- **vocab** (*dict*) – A dictionary record the vocabulary information.
- **side_data** (*dict*) – A dictionary record the side data.

build_model()
build model

forward(*batch*, *mode*)
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.

Module contents

Module contents

5.1.3 crslab.model.policy package

Subpackages

crslab.model.policy.conv_bert package

Submodules

Conv_BERT

References

Zhou, Kun, et al. "Towards Topic-Guided Conversational Recommender System." in COLING 2020.

---

class crslab.model.policy.conv_bert.conv_bert.**ConvBERTModel**(*opt*, *device*, *vocab*, *side_data*)

Bases: *crslab.model.base.BaseModel*

topic_class_num
the number of topic.

Parameters

- **opt** (*dict*) – A dictionary record the hyper parameters.
- **device** (*torch.device*) – A variable indicating which device to place the data and model.
- **vocab** (*dict*) – A dictionary record the vocabulary information.
- **side_data** (*dict*) – A dictionary record the side data.

build_model(**args*, ***kwargs*)
build model

forward(*batch*, *mode*)
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.

Module contents

crslab.model.policy.mgcg package

Submodules

MGCG

References

Liu, Zeming, et al. "Towards Conversational Recommendation over Multi-Type Dialogs." in ACL 2020.

class crslab.model.policy.mgcg.mgcg.**MGCGModel**(*opt*, *device*, *vocab*, *side_data*)
Bases: *crslab.model.base.BaseModel*

topic_class_num
A integer indicating the number of topic.

vocab_size
A integer indicating the size of vocabulary.

embedding_dim
A integer indicating the dimension of embedding layer.

hidden_size
A integer indicating the size of hidden state.

num_layers
A integer indicating the number of layers in GRU.

dropout_hidden
A float indicating the dropout rate of hidden state.

n_sent
A integer indicating sequence length in user profile.

Parameters

- **opt** (*dict*) – A dictionary record the hyper parameters.
- **device** (*torch.device*) – A variable indicating which device to place the data and model.
- **vocab** (*dict*) – A dictionary record the vocabulary information.
- **side_data** (*dict*) – A dictionary record the side data.

build_model(**args*, ***kwargs*)
build model

forward(*batch*, *mode*)
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.

---

get_length(*input*)

Module contents

crslab.model.policy.pmi package

Submodules

PMI

class crslab.model.policy.pmi.pmi.**PMIModel**(*opt*, *device*, *vocab*, *side_data*)
Bases: *crslab.model.base.BaseModel*

topic_class_num
A integer indicating the number of topic.

pad_topic
A integer indicating the id of topic padding.

Parameters

- **opt** (*dict*) – A dictionary record the hyper parameters.
- **device** (*torch.device*) – A variable indicating which device to place the data and model.
- **vocab** (*dict*) – A dictionary record the vocabulary information.
- **side_data** (*dict*) – A dictionary record the side data.

build_model(**args*, ***kwargs*)
build model

forward(*batch*, *mode*)
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.

Module contents

crslab.model.policy.profile_bert package

Submodules

Profile_BERT

References

Zhou, Kun, et al. "Towards Topic-Guided Conversational Recommender System." in COLING 2020.

class crslab.model.policy.profile_bert.profile_bert.**ProfileBERTModel**(*opt*, *device*, *vocab*, *side_data*)

Bases: *crslab.model.base.BaseModel*

topic_class_num
A integer indicating the number of topic.

n_sent
A integer indicating sequence length in user profile.

Parameters

- **opt** (*dict*) – A dictionary record the hyper parameters.
- **device** (*torch.device*) – A variable indicating which device to place the data and model.
- **vocab** (*dict*) – A dictionary record the vocabulary information.
- **side_data** (*dict*) – A dictionary record the side data.

build_model(**args*, ***kwargs*)
build model

forward(*batch*, *mode*)
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.

Module contents

crslab.model.policy.topic_bert package

Submodules

Topic_BERT

References

Zhou, Kun, et al. "Towards Topic-Guided Conversational Recommender System." in COLING 2020.

class crslab.model.policy.topic_bert.topic_bert.**TopicBERTModel**(*opt*, *device*, *vocab*, *side_data*)

Bases: *crslab.model.base.BaseModel*

topic_class_num
A integer indicating the number of topic.

Parameters

- **opt** (*dict*) – A dictionary record the hyper parameters.
- **device** (*torch.device*) – A variable indicating which device to place the data and model.
- **vocab** (*dict*) – A dictionary record the vocabulary information.
- **side_data** (*dict*) – A dictionary record the side data.

build_model(**args*, ***kwargs*)
build model

forward(*batch*, *mode*)
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.

Module contents

Module contents

5.1.4 crslab.model.recommendation package

Subpackages

crslab.model.recommendation.bert package

Submodules

BERT

References

Devlin, Jacob, et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." in NAACL 2019.

class crslab.model.recommendation.bert.bert.**BERTModel**(*opt*, *device*, *vocab*, *side_data*)
Bases: *crslab.model.base.BaseModel*

item_size
A integer indicating the number of items.

Parameters

- **opt** (*dict*) – A dictionary record the hyper parameters.
- **device** (*torch.device*) – A variable indicating which device to place the data and model.
- **vocab** (*dict*) – A dictionary record the vocabulary information.
- **side_data** (*dict*) – A dictionary record the side data.

build_model()
build model

forward(*batch*, *mode='train'*)
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.

Module contents

crslab.model.recommendation.gru4rec package

Submodules

GRU4REC

References

Hidasi, Balázs, et al. "Session-Based Recommendations with Recurrent Neural Networks." in ICLR 2016.

class crslab.model.recommendation.gru4rec.gru4rec.**GRU4RECModel**(*opt*, *device*, *vocab*, *side_data*)

Bases: *crslab.model.base.BaseModel*

item_size
A integer indicating the number of items.

hidden_size
A integer indicating the hidden state size in GRU.

num_layers
A integer indicating the number of GRU layers.

dropout_hidden
A float indicating the dropout rate to dropout hidden state.

dropout_input
A integer indicating if we dropout the input of model.

embedding_dim
A integer indicating the dimension of item embedding.

batch_size
A integer indicating the batch size.

Parameters

- **opt** (*dict*) – A dictionary record the hyper parameters.
- **device** (*torch.device*) – A variable indicating which device to place the data and model.
- **vocab** (*dict*) – A dictionary record the vocabulary information.
- **side_data** (*dict*) – A dictionary record the side data.

build_model()
build model

cross_entropy(*seq_out*, *pos_ids*, *neg_ids*, *input_mask*)

forward(*batch*, *mode*)

Parameters

- **input_ids** – padding in left, [pad, pad, id1, id2, . . . , idn]
- **target_ids** – padding in left, [pad, pad, id2, id3, . . . , y]

reconstruct_input(*input_ids*)
convert the padding from left to right

Module contents

crslab.model.recommendation.popularity package

Submodules

Popularity

class crslab.model.recommendation.popularity.popularity.**PopularityModel**(*opt*, *device*, *vocab*, *side_data*)

Bases: *crslab.model.base.BaseModel*

item_size
A integer indicating the number of items.

Parameters

- **opt** (*dict*) – A dictionary record the hyper parameters.
- **device** (*torch.device*) – A variable indicating which device to place the data and model.
- **vocab** (*dict*) – A dictionary record the vocabulary information.
- **side_data** (*dict*) – A dictionary record the side data.

build_model()
build model

forward(*batch*, *mode*)
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.

Module contents

crslab.model.recommendation.sasrec package

Submodules

class crslab.model.recommendation.sasrec.modules.**Embeddings**(*item_size*, *hidden_size*, *max_seq_length*, *hidden_dropout_prob*)

Bases: `torch.nn.modules.module.Module`

Construct the embeddings from item, position, attribute.

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*input_ids*)
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.

---

class crslab.model.recommendation.sasrec.modules.**Encoder**(*num_hidden_layers*, *hidden_size*, *num_attention_heads*, *hidden_dropout_prob*, *hidden_act*, *attention_probs_dropout_prob*)

Bases: `torch.nn.modules.module.Module`

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*hidden_states*, *attention_mask*, *output_all_encoded_layers=True*)

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.

---

class crslab.model.recommendation.sasrec.modules.**Intermediate**(*hidden_size*, *hidden_act*, *hidden_dropout_prob*)

Bases: `torch.nn.modules.module.Module`

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*input_tensor*)

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.

---

class crslab.model.recommendation.sasrec.modules.**Layer**(*hidden_size*, *num_attention_heads*, *hidden_dropout_prob*, *hidden_act*, *attention_probs_dropout_prob*)

Bases: `torch.nn.modules.module.Module`

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*hidden_states*, *attention_mask*)

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.

class crslab.model.recommendation.sasrec.modules.**LayerNorm**(*hidden_size*, *eps=1e-12*)

Bases: `torch.nn.modules.module.Module`

Construct a layernorm module in the TF style (epsilon inside the square root).

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.

class crslab.model.recommendation.sasrec.modules.**SASRec**(*hidden_dropout_prob*, *device*, *initializer_range*, *hidden_size*, *max_seq_length*, *item_size*, *num_attention_heads*, *attention_probs_dropout_prob*, *hidden_act*, *num_hidden_layers*)

Bases: `torch.nn.modules.module.Module`

Initializes internal Module state, shared by both nn.Module and ScriptModule.

build_model()

compute_loss(*y_pred*, *y*, *subset='test'*)

cross_entropy(*seq_out*, *pos_ids*, *neg_ids*)

forward(*input_ids*, *attention_mask=None*, *output_all_encoded_layers=True*)

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.

init_model()

init_sas_weights(*module*)

Initialize the weights.

load_model(*path*)

save_model(*file_name*)

class crslab.model.recommendation.sasrec.modules.**SelfAttention**(*hidden_size*, *num_attention_heads*, *hidden_dropout_prob*, *attention_probs_dropout_prob*)

Bases: torch.nn.modules.module.Module

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*input_tensor*, *attention_mask*)

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.

transpose_for_scores(*x*)

Parameters **x** – (bs, seq_len, all_head_size)

Returns x.permute(0, 2, 1, 3), (bs, num_heads, seq_len, head_size)

crslab.model.recommendation.sasrec.modules.**gelu**(*x*)

Implementation of the gelu activation function.

For information: OpenAI GPT's gelu is slightly different (and gives slightly different results): 0.5 * x * (1 + torch.tanh(math.sqrt(2 / math.pi) * (x + 0.044715 * torch.pow(x, 3)))) Also see https://arxiv.org/abs/1606.08415

crslab.model.recommendation.sasrec.modules.**swish**(*x*)

SASREC

References

Kang, Wang-Cheng, and Julian McAuley. "Self-attentive sequential recommendation." in ICDM 2018.

class crslab.model.recommendation.sasrec.sasrec.**SASRECModel**(*opt*, *device*, *vocab*, *side_data*)

Bases: *crslab.model.base.BaseModel*

hidden_dropout_prob
A float indicating the dropout rate to dropout hidden state in SASRec.

initializer_range
A float indicating the range of parameters initiation in SASRec.

hidden_size
A integer indicating the size of hidden state in SASRec.

max_seq_length
A integer indicating the max interaction history length.

item_size
A integer indicating the number of items.

num_attention_heads
A integer indicating the head number in SASRec.

attention_probs_dropout_prob
A float indicating the dropout rate in attention layers.

hidden_act
A string indicating the activation function type in SASRec.

num_hidden_layers
A integer indicating the number of hidden layers in SASRec.

Parameters

- **opt** (*dict*) – A dictionary record the hyper parameters.
- **device** (*torch.device*) – A variable indicating which device to place the data and model.
- **vocab** (*dict*) – A dictionary record the vocabulary information.
- **side_data** (*dict*) – A dictionary record the side data.

build_model()
build model

forward(*batch*, *mode*)
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.

Module contents

crslab.model.recommendation.textcnn package

Submodules

TextCNN

References

Kim, Yoon. "Convolutional Neural Networks for Sentence Classification." in EMNLP 2014.

---

class crslab.model.recommendation.textcnn.textcnn.**TextCNNModel**(*opt*, *device*, *vocab*, *side_data*)

Bases: *crslab.model.base.BaseModel*

movie_num
A integer indicating the number of items.

num_filters
A string indicating the number of filter in CNN.

embed
A integer indicating the size of embedding layer.

filter_sizes
A string indicating the size of filter in CNN.

dropout
A float indicating the dropout rate.

Parameters

- **opt** (*dict*) – A dictionary record the hyper parameters.
- **device** (*torch.device*) – A variable indicating which device to place the data and model.
- **vocab** (*dict*) – A dictionary record the vocabulary information.
- **side_data** (*dict*) – A dictionary record the side data.

build_model()
build model

conv_and_pool(*x*, *conv*)

forward(*batch*, *mode*)
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.

Module contents

Module contents

5.1.5 crslab.model.utils package

Subpackages

crslab.model.utils.modules package

Submodules

class crslab.model.utils.modules.attention.**SelfAttentionBatch**(*dim*, *da*, *alpha=0.2*, *dropout=0.5*)

Bases: `torch.nn.modules.module.Module`

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*h*)
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.

---

class crslab.model.utils.modules.attention.**SelfAttentionSeq**(*dim*, *da*, *alpha=0.2*, *dropout=0.5*)

Bases: `torch.nn.modules.module.Module`

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*h*, *mask=None*, *return_logits=False*)
For the padding tokens, its corresponding mask is True if mask==[1, 1, 1, ...]

class crslab.model.utils.modules.transformer.**MultiHeadAttention**(*n_heads*, *dim*, *dropout=0.0*)

Bases: `torch.nn.modules.module.Module`

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*query*, *key=None*, *value=None*, *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.

---

class crslab.model.utils.modules.transformer.**TransformerDecoder**(*n_heads*, *n_layers*, *embedding_size*, *ffn_size*, *vocabulary_size*, *embedding=None*, *dropout=0.0*, *attention_dropout=0.0*, *relu_dropout=0.0*, *embeddings_scale=True*, *learn_positional_embeddings=False*, *padding_idx=None*, *n_positions=1024*)

Bases: `torch.nn.modules.module.Module`

Transformer Decoder layer.

Parameters

- **n_heads** (*int*) – the number of multihead attention heads.
- **n_layers** (*int*) – number of transformer layers.

- **embedding_size** (*int*) – the embedding sizes. Must be a multiple of n_heads.
- **ffn_size** (*int*) – the size of the hidden layer in the FFN
- **embedding** – an embedding matrix for the bottom layer of the transformer. If none, one is created for this encoder.
- **dropout** (*float*) – Dropout used around embeddings and before layer layer normalizations. This is used in Vaswani 2017 and works well on large datasets.
- **attention_dropout** (*float*) – Dropout performed after the multhead attention softmax. This is not used in Vaswani 2017.
- **padding_idx** (*int*) – Reserved padding index in the embeddings matrix.
- **learn_positional_embeddings** (*bool*) – If off, sinusoidal embeddings are used. If on, position embeddings are learned from scratch.
- **embeddings_scale** (*bool*) – Scale embeddings relative to their dimensionality. Found useful in fairseq.
- **n_positions** (*int*) – Size of the position embeddings matrix.

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*input*, *encoder_state*, *incr_state=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.

---

class crslab.model.utils.modules.transformer.**TransformerDecoderLayer**(*n_heads*, *embedding_size*, *ffn_size*, *attention_dropout=0.0*, *relu_dropout=0.0*, *dropout=0.0*)

Bases: `torch.nn.modules.module.Module`

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*x*, *encoder_output*, *encoder_mask*)
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.

---

class crslab.model.utils.modules.transformer.**TransformerEncoder**(*n_heads*, *n_layers*, *embedding_size*, *ffn_size*, *vocabulary_size*, *embedding=None*, *dropout=0.0*, *attention_dropout=0.0*, *relu_dropout=0.0*, *padding_idx=0*, *learn_positional_embeddings=False*, *embeddings_scale=False*, *reduction=True*, *n_positions=1024*)

Bases: torch.nn.modules.module.Module

Transformer encoder module.

Parameters

- **n_heads** (*int*) – the number of multihead attention heads.
- **n_layers** (*int*) – number of transformer layers.
- **embedding_size** (*int*) – the embedding sizes. Must be a multiple of n_heads.
- **ffn_size** (*int*) – the size of the hidden layer in the FFN
- **embedding** – an embedding matrix for the bottom layer of the transformer. If none, one is created for this encoder.
- **dropout** (*float*) – Dropout used around embeddings and before layer layer normalizations. This is used in Vaswani 2017 and works well on large datasets.
- **attention_dropout** (*float*) – Dropout performed after the multhead attention softmax. This is not used in Vaswani 2017.
- **relu_dropout** (*float*) – Dropout used after the ReLU in the FFN. Not used in Vaswani 2017, but used in Tensor2Tensor.
- **padding_idx** (*int*) – Reserved padding index in the embeddings matrix.
- **learn_positional_embeddings** (*bool*) – If off, sinusoidal embeddings are used. If on, position embeddings are learned from scratch.
- **embeddings_scale** (*bool*) – Scale embeddings relative to their dimensionality. Found useful in fairseq.
- **reduction** (*bool*) – If true, returns the mean vector for the entire encoding sequence.
- **n_positions** (*int*) – Size of the position embeddings matrix.

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*input*)

input data is a FloatTensor of shape [batch, seq_len, dim] mask is a ByteTensor of shape [batch, seq_len], filled with 1 when inside the sequence and 0 outside.

class crslab.model.utils.modules.transformer.**TransformerEncoderLayer**(*n_heads*, *embedding_size*, *ffn_size*, *attention_dropout=0.0*, *relu_dropout=0.0*, *dropout=0.0*)

Bases: torch.nn.modules.module.Module

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*tensor*, *mask*)

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.

---

class crslab.model.utils.modules.transformer.**TransformerFFN**(*dim*, *dim_hidden*, *relu_dropout=0.0*)

Bases: torch.nn.modules.module.Module

Initializes internal Module state, shared by both nn.Module and ScriptModule.

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.

---

crslab.model.utils.modules.transformer.**_normalize**(*tensor*, *norm_layer*)

Broadcast layer norm

crslab.model.utils.modules.transformer.**create_position_codes**(*n_pos*, *dim*, *out*)

crslab.model.utils.modules.transformer.**neginf**(*dtype*)

Returns a representable finite number near -inf for a dtype.

Module contents

Submodules

crslab.model.utils.functions.**edge_to_pyg_format**(*edge*, *type='RGCN'*)

crslab.model.utils.functions.**sort_for_packed_sequence**(*lengths: torch.Tensor*)

Parameters **lengths** – 1D array of lengths

Returns sorted_lengths (lengths in descending order), sorted_idx (indices to sort), rev_idx (indices to retrieve original order)

Module contents

5.2 Submodules

class crslab.model.base.**BaseModel**(*opt*, *device*, *dpath=None*, *resource=None*)
Bases: abc.ABC, torch.nn.modules.module.Module

Base class for all models

abstract build_model(**args*, ***kwargs*)
build model

converse(*batch*, *mode*)
calculate loss and prediction of conversation for batch under certain mode

Parameters

- **batch** (*dict or tuple*) – batch data
- **mode** (*str, optional*) – train/valid/test.

guide(*batch*, *mode*)
calculate loss and prediction of guidance for batch under certain mode

Parameters

- **batch** (*dict or tuple*) – batch data
- **mode** (*str, optional*) – train/valid/test.

recommend(*batch*, *mode*)
calculate loss and prediction of recommendation for batch under certain mode

Parameters

- **batch** (*dict or tuple*) – batch data
- **mode** (*str, optional*) – train/valid/test.

5.3 Module contents

crslab.model.**get_model**(*config*, *model_name*, *device*, *vocab*, *side_data=None*)

CHAPTER

SIX

CRSLAB.SYSTEM PACKAGE

6.1 Submodules

6.2 Module contents

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

C

INDEX

Symbols

A

B

C

D

E

F

G

N

O

P

R

S

T

U

V