MWPToolkit

Release 0.0.6

"

Apr 23, 2022

MWPTOOLKIT API:

CHAPTER

ONE

MWPTOOLKIT.CONFIG.CONFIGURATION

class mwptoolkit.config.configuration.**Config**(*model_name=None*, *dataset_name=None*, *task_type=None*, *config_dict={}*)

Bases: object

The class for loading pre-defined parameters.

Config will load the parameters from internal config file, dataset config file, model config file, config dictionary and cmd line.

The default road path of internal config file is 'mwptoolkit/config/config.json', and it's not supported to change.

The dataset config, model config and config dictionary are called the external config.

According to specific dataset and model, this class will load the dataset config from default road path 'mwptoolkit/properties/dataset/dataset_name.json' and model config from default road path 'mwptoolkit/properties/model/model_name.json'.

You can set the parameters 'model_config_path' and 'dataset_config_path' to load your own model and dataset config, but note that only json file can be loaded correctly. Config dictionary is a dict-like object. When you initialize the Config object, you can pass config dictionary through the code 'config = Config(config_dict=config_dict)'

Cmd line requires you keep the template –param_name=param_value to set any parameter you want.

If there are multiple values of the same parameter, the priority order is as following:

cmd line > external config > internal config

in external config, config dictionary > model config > dataset config.

Parameters

- **model_name** (*str*) – the model name, default is None, if it is None, config will search the parameter 'model'
- **name.** (*from the external input as the dataset*) –
- **dataset_name** (*str*) – the dataset name, default is None, if it is None, config will search the parameter 'dataset'
- **name.** –
- **task_type** (*str*) – the task type, default is None, if it is None, config will search the parameter 'task_type'
- **type.** (*from the external input as the task*) –
- **config_dict** (*dict*) – the external parameter dictionaries, default is None.

_convert_config_dict(*config_dict*)

This function convert the str parameters to their original type.

_load_cmd_line()

Read parameters from command line and convert it to str.

classmethod load_from_pretrained(*pretrained_dir*)

save_config(*trained_dir*)

to_dict()

CHAPTER

TWO

MWPTOOLKIT.DATA

2.1 mwptoolkit.data.dataloader

2.1.1 mwptoolkit.data.dataloader.abstract_dataloader

class mwptoolkit.data.dataloader.abstract_dataloader.**AbstractDataLoader**(*config*, *dataset*)

Bases: object

abstract dataloader

the base class of dataloader class

Parameters

- **config** –
- **dataset** –

expected that config includes these parameters below:

model (str): model name.

equation_fix (str): [infix | postfix | prefix], convert equation to specified format.

train_batch_size (int): the training batch size.

test_batch_size (int): the testing batch size.

symbol_for_tree (bool): build output symbols for tree or not.

share_vocab (bool): encoder and decoder of the model share the same vocabulary, often seen in Seq2Seq models.

max_len (int|None): max input length.

max_equ_len (int|None): max output length.

add_sos (bool): add sos token at the head of input sequence.

add_eos (bool): add eos token at the tail of input sequence.

device (torch.device):

convert_idx_2_symbol(*equation_idx: List[int]*)

convert symbol index of equation to symbol. :param equation_idx: :return:

convert_idx_2_word(*sentence_idx: List[int]*)

convert token index of input sequence to token. :param sentence_idx: :return:

convert_symbol_2_idx(*equation: List[str]*)

convert symbol of equation to index. :param equation: :return:

convert_word_2_idx(*sentence: List[str]*)

convert token of input sequence to index. :param sentence: List[str] :return:

init_batches()

initialize batches.

load_data()

load data.

load_next_batch()

load data.

2.1.2 mwptoolkit.data.dataloader.dataloader_ept

class mwptoolkit.data.dataloader.dataloader_ept.**DataLoaderEPT**(*config:* mwptoolkit.config.configuration.Config, *dataset:* mwptoolkit.data.dataset.dataset_ept.DatasetEPT)

Bases: *mwptoolkit.data.dataloader.template_dataloader.TemplateDataLoader*

dataloader class for deep-learning model EPT

Parameters

- **config** –
- **dataset** –

expected that config includes these parameters below:

dataset (str): dataset name.

pretrained_model_path (str): road path of pretrained model.

decoder (str): decoder module name.

model (str): model name.

equation_fix (str): [infix | postfix | prefix], convert equation to specified format.

train_batch_size (int): the training batch size.

test_batch_size (int): the testing batch size.

symbol_for_tree (bool): build output symbols for tree or not.

share_vocab (bool): encoder and decoder of the model share the same vocabulary, often seen in Seq2Seq models.

max_len (int|None): max input length.

add_sos (bool): add sos token at the head of input sequence.

add_eos (bool): add eos token at the tail of input sequence.

build_batch_for_predict(*batch_data: List[dict]*)

2.1.3 mwptoolkit.data.dataloader.dataloader_hms

class mwptoolkit.data.dataloader.dataloader_hms.**DataLoaderHMS**(*config:* mwptoolkit.config.configuration.Config, *dataset:* mwptoolkit.data.dataset.dataset_hms.DatasetHMS)

Bases: *mwptoolkit.data.dataloader.template_dataloader.TemplateDataLoader*

Parameters

- **config** –
- **dataset** –

expected that config includes these parameters below:

model (str): model name.

equation_fix (str): [infix | postfix | prefix], convert equation to specified format.

train_batch_size (int): the training batch size.

test_batch_size (int): the testing batch size.

symbol_for_tree (bool): build output symbols for tree or not.

share_vocab (bool): encoder and decoder of the model share the same vocabulary, often seen in Seq2Seq models.

max_len (int|None): max input length.

max_equ_len (int|None): max output length.

add_sos (bool): add sos token at the head of input sequence.

add_eos (bool): add eos token at the tail of input sequence.

device (torch.device):

build_batch_for_predict(*batch_data: List[dict]*)

mwptoolkit.data.dataloader.dataloader_hms.**get_num_mask**(*num_size_batch*, *generate_nums*)

2.1.4 mwptoolkit.data.dataloader.dataloader_multiencdec

class mwptoolkit.data.dataloader.dataloader_multiencdec.**DataLoaderMultiEncDec**(*config:* mwptoolkit.config.configuration.Confi *dataset:* mwptoolkit.data.dataset.dataset_multi

Bases: *mwptoolkit.data.dataloader.template_dataloader.TemplateDataLoader*

dataloader class for deep-learning model MultiE&D

Parameters

- **config** –
- **dataset** –

expected that config includes these parameters below:

model (str): model name.

equation_fix (str): [infix | postfix | prefix], convert equation to specified format.

train_batch_size (int): the training batch size.

test_batch_size (int): the testing batch size.

symbol_for_tree (bool): build output symbols for tree or not.

share_vocab (bool): encoder and decoder of the model share the same vocabulary, often seen in Seq2Seq models.

max_len (int|None): max input length.

max_equ_len (int|None): max output length.

add_sos (bool): add sos token at the head of input sequence.

add_eos (bool): add eos token at the tail of input sequence.

device (torch.device):

build_batch_for_predict(*batch_data: List[dict]*)

get_parse_graph_batch(*input_length*, *parse_tree_batch*)

num_order_processed(*num_list*)

2.1.5 mwptoolkit.data.dataloader.multi_equation_dataloader

class mwptoolkit.data.dataloader.multi_equation_dataloader.**MultiEquationDataLoader**(*config:* mwp-toolkit.config.configuratio *dataset:* mwp-toolkit.data.dataset.multi_e

Bases: *mwptoolkit.data.dataloader.abstract_dataloader.AbstractDataLoader*

multiple-equation dataloader

Parameters

- **config** –
- **dataset** –

expected that config includes these parameters below:

model (str): model name.

equation_fix (str): [infix | postfix | prefix], convert equation to specified format.

train_batch_size (int): the training batch size.

test_batch_size (int): the testing batch size.

symbol_for_tree (bool): build output symbols for tree or not.

share_vocab (bool): encoder and decoder of the model share the same vocabulary, often seen in Seq2Seq models.

max_len (int|None): max input length.

max_equ_len (int|None): max output length.

add_sos (bool): add sos token at the head of input sequence.

add_eos (bool): add eos token at the tail of input sequence.

device (torch.device):

build_batch_for_predict(*batch_data: List[dict]*)

init_batches()

Initialize batches of trainset, validset and testset. :return: None

load_data(*type: str*)

Load batches, return every batch data in a generator object.

Parameters **type** – [train | valid | test], data type.

Returns Generator[dict], batches

load_next_batch(*type: str*) → dict

Return next batch data :param type: [train | valid | test], data type. :return: batch data

mwptoolkit.data.dataloader.multi_equation_dataloader.**get_num_mask**(*num_size_batch*, *generate_nums*)

2.1.6 mwptoolkit.data.dataloader.pretrain_dataloader

class mwptoolkit.data.dataloader.pretrain_dataloader.**PretrainDataLoader**(*config: mwptoolkit.config.configuration.Config*, *dataset: mwptoolkit.data.dataset.pretrain_dataset.Pretr*

Bases: *mwptoolkit.data.dataloader.abstract_dataloader.AbstractDataLoader*

dataloader class for pre-train model.

Parameters

- **config** –
- **dataset** –

expected that config includes these parameters below:

model (str): model name.

equation_fix (str): [infix | postfix | prefix], convert equation to specified format.

train_batch_size (int): the training batch size.

test_batch_size (int): the testing batch size.

symbol_for_tree (bool): build output symbols for tree or not.

share_vocab (bool): encoder and decoder of the model share the same vocabulary, often seen in Seq2Seq models.

max_len (int|None): max input length.

max_equ_len (int|None): max output length.

add_sos (bool): add sos token at the head of input sequence.

add_eos (bool): add eos token at the tail of input sequence.

device (torch.device):

build_batch_for_predict(*batch_data: List[dict]*)

init_batches()

Initialize batches of trainset, validset and testset. :return: None

load_data(*type: str*)

Load batches, return every batch data in a generator object.

Parameters **type** – [train | valid | test], data type.

Returns Generator[dict], batches

load_next_batch(*type: str*) → dict

Return next batch data :param type: [train | valid | test], data type. :return: batch data

mwptoolkit.data.dataloader.pretrain_dataloader.**get_num_mask**(*num_size_batch*, *generate_nums*)

2.1.7 mwptoolkit.data.dataloader.single_equation_dataloader

class mwptoolkit.data.dataloader.single_equation_dataloader.**SingleEquationDataLoader**(*config:* mwp-toolkit.config.configurat *dataset:* mwp-toolkit.data.dataset.singl

Bases: *mwptoolkit.data.dataloader.abstract_dataloader.AbstractDataLoader*

single-equation dataloader

Parameters

- **config** –
- **dataset** –

expected that config includes these parameters below:

model (str): model name.

equation_fix (str): [infix | postfix | prefix], convert equation to specified format.

train_batch_size (int): the training batch size.

test_batch_size (int): the testing batch size.

symbol_for_tree (bool): build output symbols for tree or not.

share_vocab (bool): encoder and decoder of the model share the same vocabulary, often seen in Seq2Seq models.

max_len (int|None): max input length.

max_equ_len (int|None): max output length.

add_sos (bool): add sos token at the head of input sequence.

add_eos (bool): add eos token at the tail of input sequence.

device (torch.device):

build_batch_for_predict(*batch_data: List[dict]*)

init_batches()

Initialize batches of trainset, validset and testset. :return: None

load_data(*type: str*)

Load batches, return every batch data in a generator object.

Parameters **type** – [train | valid | test], data type.

Returns Generator[dict], batches

load_next_batch(*type: str*) → dict

Return next batch data :param type: [train | valid | test], data type. :return: batch data

mwptoolkit.data.dataloader.single_equation_dataloader.**get_num_mask**(*num_size_batch*, *generate_nums*)

2.1.8 mwptoolkit.data.dataloader.template_dataloader

class mwptoolkit.data.dataloader.template_dataloader.**TemplateDataLoader**(*config*, *dataset*)

Bases: *mwptoolkit.data.dataloader.abstract_dataloader.AbstractDataLoader*

template dataloader.

you need implement:

TemplateDataLoader.__init_batches()

We replace abstract method TemplateDataLoader.load_batch() with TemplateDataLoader.__init_batches() after version 0.0.5 . Their functions are similar.

Parameters

- **config** –
- **dataset** –

expected that config includes these parameters below:

model (str): model name.

equation_fix (str): [infix | postfix | prefix], convert equation to specified format.

train_batch_size (int): the training batch size.

test_batch_size (int): the testing batch size.

symbol_for_tree (bool): build output symbols for tree or not.

share_vocab (bool): encoder and decoder of the model share the same vocabulary, often seen in Seq2Seq models.

max_len (int|None): max input length.

max_equ_len (int|None): max output length.

add_sos (bool): add sos token at the head of input sequence.

add_eos (bool): add eos token at the tail of input sequence.

device (torch.device):

init_batches()

Initialize batches of trainset, validset and testset. :return: None

load_data(*type: str*)

Load batches, return every batch data in a generator object.

Parameters **type** – [train | valid | test], data type.

Returns Generator[dict], batches

load_next_batch(*type: str*)

Return next batch data :param type: [train | valid | test], data type. :return: batch data

2.2 mwptoolkit.data.dataset

2.2.1 mwptoolkit.data.dataset.abstract_dataset

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

Bases: object

abstract dataset

the base class of dataset class

Parameters **config** (mwptoolkit.config.configuration.Config) –

expected that config includes these parameters below:

model (str): model name.

dataset (str): dataset name.

equation_fix (str): [infix | postfix | prefix], convert equation to specified format.

dataset_dir or dataset_path (str): the road path of dataset folder.

language (str): a property of dataset, the language of dataset.

single (bool): a property of dataset, the equation of dataset is single or not.

linear (bool): a property of dataset, the equation of dataset is linear or not.

source_equation_fix (str): [infix | postfix | prefix], a property of dataset, the source format of equation of dataset.

rebuild (bool): when loading additional dataset information, this can decide to build information anew or load information built before.

validset_divide (bool): whether to split validset. if True, the dataset is split to trainset-validset-testset. if False, the dataset is split to trainset-testset.

mask_symbol (str): [NUM | number], the symbol to mask numbers in equation.

min_word_keep (int): in dataset, words that count greater than the value, will be kept in input vocabulary.

min_generate_keep (int): generate number that count greater than the value, will be kept in output symbols.

symbol_for_tree (bool): build output symbols for tree or not.

share_vocab (bool): encoder and decoder of the model share the same vocabulary, often seen in Seq2Seq models.

k_fold (int|None): if it's an integer, it indicates to run k-fold cross validation. if it's None, it indicates to run trainset-validset-testset split.

read_local_folds (bool): when running k-fold cross validation, if True, then loading split folds from dataset folder. if False, randomly split folds.

shuffle (bool): whether to shuffle trainset before training.

device (torch.device):

resume_training or resume (bool):

_load_dataset()

read dataset from files

_load_fold_dataset()

read one fold of dataset from file.

cross_validation_load(*k_fold*, *start_fold_t=None*)

dataset load for cross validation

Build folds for cross validation. Choose one of folds for testset and other folds for trainset.

Parameters

- **k_fold** (*int*) – the number of folds, also the cross validation parameter k.
- **start_fold_t** (*int*) – default None, training start from the training of t-th time.

Returns Generator including current training index of cross validation.

dataset_load()

dataset process and build vocab.

when running k-fold setting, this function required to call once per fold.

en_rule1_process(*k*)

en_rule2_process()

fix_process(*fix*)

equation infix/postfix/prefix process.

Parameters **fix** (*function*) – a function to make infix, postfix, prefix or None

classmethod load_from_pretrained(*pretrained_dir*)

operator_mask_process()

operator mask process of equation.

parameters_to_dict()

return the parameters of dataset as format of dict. :return:

reset_dataset()

save_dataset(*trained_dir*)

2.2.2 mwptoolkit.data.dataset.dataset_ept

class mwptoolkit.data.dataset.dataset_ept.**DatasetEPT**(*config*)

Bases: *mwptoolkit.data.dataset.template_dataset.TemplateDataset*

dataset class for deep-learning model EPT.

Parameters **config** (mwptoolkit.config.configuration.Config) –

expected that config includes these parameters below:

task_type (str): [single_equation | multi_equation], the type of task.

pretrained_model or transformers_pretrained_model (str|None): road path or name of pretrained model.

decoder (str): decoder module name.

model (str): model name.

dataset (str): dataset name.

equation_fix (str): [infix | postfix | prefix], convert equation to specified format.

dataset_dir or dataset_path (str): the road path of dataset folder.

language (str): a property of dataset, the language of dataset.

single (bool): a property of dataset, the equation of dataset is single or not.

linear (bool): a property of dataset, the equation of dataset is linear or not.

source_equation_fix (str): [infix | postfix | prefix], a property of dataset, the source format of equation of dataset.

rebuild (bool): when loading additional dataset information, this can decide to build information anew or load information built before.

validset_divide (bool): whether to split validset. if True, the dataset is split to trainset-validset-testset. if False, the dataset is split to trainset-testset.

mask_symbol (str): [NUM | number], the symbol to mask numbers in equation.

min_word_keep (int): in dataset, words that count greater than the value, will be kept in input vocabulary.

min_generate_keep (int): generate number that count greater than the value, will be kept in output symbols.

symbol_for_tree (bool): build output symbols for tree or not.

share_vocab (bool): encoder and decoder of the model share the same vocabulary, often seen in Seq2Seq models.

k_fold (int|None): if it's an integer, it indicates to run k-fold cross validation. if it's None, it indicates to run trainset-validset-testset split.

read_local_folds (bool): when running k-fold cross validation, if True, then loading split folds from dataset folder. if False, randomly split folds.

shuffle (bool): whether to shuffle trainset before training.

device (torch.device):

resume_training or resume (bool):

`get_vocab_size()`

Returns the length of input vocabulary and output symbols

Return type (tuple(int, int))

`classmethod load_from_pretrained(`*pretrained_dir: str*, *resume_training=False*`)`

load dataset parameters from file.

Parameters

- **pretrained_dir** – (str) folder which saved the parameter file
- **resume_training** – (bool) load parameter for resuming training or not.

Returns an instantiated object

`save_dataset(`*save_dir: str*`)`

save dataset parameters to file.

Parameters **save_dir** – (str) folder which saves the parameter file

Returns

2.2.3 mwptoolkit.data.dataset.dataset_hms

class mwptoolkit.data.dataset.dataset_hms.**DatasetHMS**(*config*)

Bases: *mwptoolkit.data.dataset.template_dataset.TemplateDataset*

dataset class for deep-learning model HMS

Parameters **config** (mwptoolkit.config.configuration.Config) –

expected that config includes these parameters below:

rule1 (bool): convert equation according to rule 1.

rule2 (bool): convert equation according to rule 2.

parse_tree_file_name (str|None): the name of the file to save parse tree information.

model (str): model name.

dataset (str): dataset name.

equation_fix (str): [infix | postfix | prefix], convert equation to specified format.

dataset_dir or dataset_path (str): the road path of dataset folder.

language (str): a property of dataset, the language of dataset.

single (bool): a property of dataset, the equation of dataset is single or not.

linear (bool): a property of dataset, the equation of dataset is linear or not.

source_equation_fix (str): [infix | postfix | prefix], a property of dataset, the source format of equation of dataset.

rebuild (bool): when loading additional dataset information, this can decide to build information anew or load information built before.

validset_divide (bool): whether to split validset. if True, the dataset is split to trainset-validset-testset. if False, the dataset is split to trainset-testset.

mask_symbol (str): [NUM | number], the symbol to mask numbers in equation.

min_word_keep (int): in dataset, words that count greater than the value, will be kept in input vocabulary.

min_generate_keep (int): generate number that count greater than the value, will be kept in output symbols.

symbol_for_tree (bool): build output symbols for tree or not.

share_vocab (bool): encoder and decoder of the model share the same vocabulary, often seen in Seq2Seq models.

k_fold (int|None): if it's an integer, it indicates to run k-fold cross validation. if it's None, it indicates to run trainset-validset-testset split.

read_local_folds (bool): when running k-fold cross validation, if True, then loading split folds from dataset folder. if False, randomly split folds.

shuffle (bool): whether to shuffle trainset before training.

device (torch.device):

resume_training or resume (bool):

get_vocab_size()

Returns the length of input vocabulary and output symbols

Return type (tuple(int, int))

classmethod load_from_pretrained(*pretrained_dir: str*, *resume_training=False*)

load dataset parameters from file.

Parameters

- **pretrained_dir** – (str) folder which saved the parameter file
- **resume_training** – (bool) load parameter for resuming training or not.

Returns an instantiated object

save_dataset(*save_dir: str*)

save dataset parameters to file.

Parameters **save_dir** – (str) folder which saves the parameter file

Returns

2.2.4 mwptoolkit.data.dataset.dataset_multiencddec

class mwptoolkit.data.dataset.dataset_multiencdec.**DatasetMultiEncDec**(*config*)

Bases: *mwptoolkit.data.dataset.template_dataset.TemplateDataset*

dataset class for deep-learning model MultiE&D

Parameters **config** (mwptoolkit.config.configuration.Config) –

expected that config includes these parameters below:

task_type (str): [single_equation | multi_equation], the type of task.

parse_tree_file_name (str|None): the name of the file to save parse tree information.

ltp_model_dir or ltp_model_path (str|None): the road path of ltp model.

model (str): model name.

dataset (str): dataset name.

equation_fix (str): [infix | postfix | prefix], convert equation to specified format.

dataset_dir or dataset_path (str): the road path of dataset folder.

language (str): a property of dataset, the language of dataset.

single (bool): a property of dataset, the equation of dataset is single or not.

linear (bool): a property of dataset, the equation of dataset is linear or not.

source_equation_fix (str): [infix | postfix | prefix], a property of dataset, the source format of equation of dataset.

rebuild (bool): when loading additional dataset information, this can decide to build information anew or load information built before.

validset_divide (bool): whether to split validset. if True, the dataset is split to trainset-validset-testset. if False, the dataset is split to trainset-testset.

mask_symbol (str): [NUM | number], the symbol to mask numbers in equation.

min_word_keep (int): in dataset, words that count greater than the value, will be kept in input vocabulary.

min_generate_keep (int): generate number that count greater than the value, will be kept in output symbols.

symbol_for_tree (bool): build output symbols for tree or not.

share_vocab (bool): encoder and decoder of the model share the same vocabulary, often seen in Seq2Seq models.

k_fold (int|None): if it's an integer, it indicates to run k-fold cross validation. if it's None, it indicates to run trainset-validset-testset split.

read_local_folds (bool): when running k-fold cross validation, if True, then loading split folds from dataset folder. if False, randomly split folds.

shuffle (bool): whether to shuffle trainset before training.

device (torch.device):

resume_training or resume (bool):

build_pos_to_file_with_pyltp(*path*)

build_pos_to_file_with_stanza(*path*)

classmethod load_from_pretrained(*pretrained_dir: str*, *resume_training=False*)

load dataset parameters from file.

Parameters

- **pretrained_dir** – (str) folder which saved the parameter file
- **resume_training** – (bool) load parameter for resuming training or not.

Returns an instantiated object

read_pos_from_file(*path*)

save_dataset(*save_dir: str*)

save dataset parameters to file.

Parameters **save_dir** – (str) folder which saves the parameter file

Returns

2.2.5 mwptoolkit.data.dataset.multi_equation_dataset

class mwptoolkit.data.dataset.multi_equation_dataset.**MultiEquationDataset**(*config*)

Bases: *mwptoolkit.data.dataset.abstract_dataset.AbstractDataset*

multiple-equation dataset.

Parameters **config** (mwptoolkit.config.configuration.Config) –

expected that config includes these parameters below:

rule1 (bool): convert equation according to rule 1.

rule2 (bool): convert equation according to rule 2.

parse_tree_file_name (str|None): the name of the file to save parse tree information.

model (str): model name.

dataset (str): dataset name.

equation_fix (str): [infix | postfix | prefix], convert equation to specified format.

dataset_dir or dataset_path (str): the road path of dataset folder.

language (str): a property of dataset, the language of dataset.

single (bool): a property of dataset, the equation of dataset is single or not.

linear (bool): a property of dataset, the equation of dataset is linear or not.

source_equation_fix (str): [infix | postfix | prefix], a property of dataset, the source format of equation of dataset.

rebuild (bool): when loading additional dataset information, this can decide to build information anew or load information built before.

validset_divide (bool): whether to split validset. if True, the dataset is split to trainset-validset-testset. if False, the dataset is split to trainset-testset.

mask_symbol (str): [NUM | number], the symbol to mask numbers in equation.

min_word_keep (int): in dataset, words that count greater than the value, will be kept in input vocabulary.

min_generate_keep (int): generate number that count greater than the value, will be kept in output symbols.

symbol_for_tree (bool): build output symbols for tree or not.

share_vocab (bool): encoder and decoder of the model share the same vocabulary, often seen in Seq2Seq models.

k_fold (int|None): if it's an integer, it indicates to run k-fold cross validation. if it's None, it indicates to run trainset-validset-testset split.

read_local_folds (bool): when running k-fold cross validation, if True, then loading split folds from dataset folder. if False, randomly split folds.

shuffle (bool): whether to shuffle trainset before training.

device (torch.device):

resume_training or resume (bool):

get_vocab_size()

Returns the length of input vocabulary and output symbols

Return type (tuple(int, int))

classmethod load_from_pretrained(*pretrained_dir: str*, *resume_training=False*)

load dataset parameters from file.

Parameters

- **pretrained_dir** – (str) folder which saved the parameter file
- **resume_training** – (bool) load parameter for resuming training or not.

Returns an instantiated object

save_dataset(*save_dir: str*)

save dataset parameters to file.

Parameters **save_dir** – (str) folder which saves the parameter file

Returns

2.2.6 mwptoolkit.data.dataset.pretrain_dataset

class mwptoolkit.data.dataset.pretrain_dataset.**PretrainDataset**(*config*)

Bases: *mwptoolkit.data.dataset.abstract_dataset.AbstractDataset*

dataset class for pre-train model.

Parameters **config** (mwptoolkit.config.configuration.Config) –

expected that config includes these parameters below:

task_type (str): [single_equation | multi_equation], the type of task.

embedding (str|None): embedding module name, use pre-train model as embedding module, if None, not to use pre-train model.

rule1 (bool): convert equation according to rule 1.

rule2 (bool): convert equation according to rule 2.

parse_tree_file_name (str|None): the name of the file to save parse tree information.

pretrained_model or transformers_pretrained_model (str|None): road path or name of pretrained model.

model (str): model name.

dataset (str): dataset name.

equation_fix (str): [infix | postfix | prefix], convert equation to specified format.

dataset_dir or dataset_path (str): the road path of dataset folder.

language (str): a property of dataset, the language of dataset.

single (bool): a property of dataset, the equation of dataset is single or not.

linear (bool): a property of dataset, the equation of dataset is linear or not.

source_equation_fix (str): [infix | postfix | prefix], a property of dataset, the source format of equation of dataset.

rebuild (bool): when loading additional dataset information, this can decide to build information anew or load information built before.

validset_divide (bool): whether to split validset. if True, the dataset is split to trainset-validset-testset. if False, the dataset is split to trainset-testset.

mask_symbol (str): [NUM | number], the symbol to mask numbers in equation.

min_word_keep (int): in dataset, words that count greater than the value, will be kept in input vocabulary.

min_generate_keep (int): generate number that count greater than the value, will be kept in output symbols.

symbol_for_tree (bool): build output symbols for tree or not.

share_vocab (bool): encoder and decoder of the model share the same vocabulary, often seen in Seq2Seq models.

k_fold (int|None): if it's an integer, it indicates to run k-fold cross validation. if it's None, it indicates to run trainset-validset-testset split.

read_local_folds (bool): when running k-fold cross validation, if True, then loading split folds from dataset folder. if False, randomly split folds.

shuffle (bool): whether to shuffle trainset before training.

device (torch.device):

resume_training or resume (bool):

get_vocab_size()

Returns the length of input vocabulary and output symbols

Return type (tuple(int, int))

classmethod load_from_pretrained(*pretrained_dir: str*, *resume_training=False*)

load dataset parameters from file.

Parameters

- **pretrained_dir** – (str) folder which saved the parameter file
- **resume_training** – (bool) load parameter for resuming training or not.

Returns an instantiated object

save_dataset(*save_dir: str*)

save dataset parameters to file.

Parameters **save_dir** – (str) folder which saves the parameter file

Returns

2.2.7 mwptoolkit.data.dataset.single_equation_dataset

class mwptoolkit.data.dataset.single_equation_dataset.**SingleEquationDataset**(*config*)

Bases: *mwptoolkit.data.dataset.abstract_dataset.AbstractDataset*

single-equation dataset

preprocess dataset when running single-equation task.

Parameters **config** (mwptoolkit.config.configuration.Config) –

expected that config includes these parameters below:

rule1 (bool): convert equation according to rule 1.

rule2 (bool): convert equation according to rule 2.

parse_tree_file_name (str|None): the name of the file to save parse tree information.

model (str): model name.

dataset (str): dataset name.

equation_fix (str): [infix | postfix | prefix], convert equation to specified format.

dataset_dir or dataset_path (str): the road path of dataset folder.

language (str): a property of dataset, the language of dataset.

single (bool): a property of dataset, the equation of dataset is single or not.

linear (bool): a property of dataset, the equation of dataset is linear or not.

source_equation_fix (str): [infix | postfix | prefix], a property of dataset, the source format of equation of dataset.

rebuild (bool): when loading additional dataset information, this can decide to build information anew or load information built before.

validset_divide (bool): whether to split validset. if True, the dataset is split to trainset-validset-testset. if False, the dataset is split to trainset-testset.

mask_symbol (str): [NUM | number], the symbol to mask numbers in equation.

min_word_keep (int): in dataset, words that count greater than the value, will be kept in input vocabulary.

min_generate_keep (int): generate number that count greater than the value, will be kept in output symbols.

symbol_for_tree (bool): build output symbols for tree or not.

share_vocab (bool): encoder and decoder of the model share the same vocabulary, often seen in Seq2Seq models.

k_fold (int|None): if it's an integer, it indicates to run k-fold cross validation. if it's None, it indicates to run trainset-validset-testset split.

read_local_folds (bool): when running k-fold cross validation, if True, then loading split folds from dataset folder. if False, randomly split folds.

shuffle (bool): whether to shuffle trainset before training.

device (torch.device):

resume_training or resume (bool):

get_vocab_size()

Returns the length of input vocabulary and output symbols

Return type (tuple(int, int))

classmethod load_from_pretrained(*pretrained_dir: str*, *resume_training=False*)

load dataset parameters from file.

Parameters

- **pretrained_dir** – (str) folder which saved the parameter file
- **resume_training** – (bool) load parameter for resuming training or not.

Returns an instantiated object

save_dataset(*save_dir: str*)

save dataset parameters to file.

Parameters **save_dir** – (str) folder which saves the parameter file

Returns

2.2.8 mwptoolkit.data.dataset.template_dataset

class mwptoolkit.data.dataset.template_dataset.**TemplateDataset**(*config*)

Bases: *mwptoolkit.data.dataset.abstract_dataset.AbstractDataset*

template dataset.

you need implement:

TemplateDataset._preprocess()

TemplateDataset._build_symbol()

TemplateDataset._build_template_symbol()

overwrite TemplateDataset._build_vocab() if necessary

Parameters **config** (mwptoolkit.config.configuration.Config) –

expected that config includes these parameters below:

model (str): model name.

dataset (str): dataset name.

equation_fix (str): [infix | postfix | prefix], convert equation to specified format.

dataset_dir or dataset_path (str): the road path of dataset folder.

language (str): a property of dataset, the language of dataset.

single (bool): a property of dataset, the equation of dataset is single or not.

linear (bool): a property of dataset, the equation of dataset is linear or not.

source_equation_fix (str): [infix | postfix | prefix], a property of dataset, the source format of equation of dataset.

rebuild (bool): when loading additional dataset information, this can decide to build information anew or load information built before.

validset_divide (bool): whether to split validset. if True, the dataset is split to trainset-validset-testset. if False, the dataset is split to trainset-testset.

mask_symbol (str): [NUM | number], the symbol to mask numbers in equation.

min_word_keep (int): in dataset, words that count greater than the value, will be kept in input vocabulary.

min_generate_keep (int): generate number that count greater than the value, will be kept in output symbols.

symbol_for_tree (bool): build output symbols for tree or not.

share_vocab (bool): encoder and decoder of the model share the same vocabulary, often seen in Seq2Seq models.

k_fold (int|None): if it's an integer, it indicates to run k-fold cross validation. if it's None, it indicates to run trainset-validset-testset split.

read_local_folds (bool): when running k-fold cross validation, if True, then loading split folds from dataset folder. if False, randomly split folds.

shuffle (bool): whether to shuffle trainset before training.

device (torch.device):

resume_training or resume (bool):

_build_symbol()

In this function, you need to implement the codes of building output vocabulary.

Specifically, you need to

1. reset the list variables TemplateDataset.out_idx2symbol, append the generating symbols into it.

you should return a dictionary object like >>> {'out_idx2symbol':out_idx2symbol}

_build_template_symbol()

In this function, you need to implement the codes of building output vocabulary for equation template.

Specifically, you need to

1. reset the list variables TemplateDataset.temp_idx2symbol, append the generating symbols into it. Also, you can do nothing in this function if you don't need template.

ou should return a dictionary object like >>> {'temp_idx2symbol':temp_idx2symbol}

_preprocess()

In this function, you need to implement the codes of data preprocessing.

Specifically, you need to

1. format input and output of every data, including trainset, validset and testset.
2. reset the list variables TemplateDataset.generate_list, TemplateDataset.operator_list and TemplateDataset.special_token_list.
3. reset the integer variables TemplateDataset.copy_nums

you should return a dictionary object like >>> {

'generate_list':generate_list, 'operator_list':operator_list, 'special_token_list':special_token_list, 'copy_nums':copy_nums

}

get_vocab_size()

classmethod load_from_pretrained(*pretrained_dir: str*, *resume_training=False*)

load dataset parameters from file.

Parameters

- **pretrained_dir** – (str) folder which saved the parameter file
- **resume_training** – (bool) load parameter for resuming training or not.

Returns an instantiated object

save_dataset(*save_dir: str*)

save dataset parameters to file.

Parameters **save_dir** – (str) folder which saves the parameter file

Returns

2.3 mwptoolkit.data.utils

mwptoolkit.data.utils.**create_dataloader**(*config*)

Create dataloader according to config

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

Returns Dataloader module

mwptoolkit.data.utils.**create_dataset**(*config*)

Create dataset according to config

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

Returns Constructed dataset.

Return type Dataset

mwptoolkit.data.utils.**get_dataloader_module**(*config:* mwptoolkit.config.configuration.Config) → Type[Union[*mwptoolkit.data.dataloader.dataloader_multiencdec.DataLoader*, *mwptoolkit.data.dataloader.dataloader_ept.DataLoaderEPT*, *mwptoolkit.data.dataloader.dataloader_hms.DataLoaderHMS*, mwptoolkit.data.dataloader.dataloader_gpt2.DataLoaderGPT2, *mwptoolkit.data.dataloader.pretrain_dataloader.PretrainDataLoader*, *mwptoolkit.data.dataloader.single_equation_dataloader.SingleEquationDataLoad*, *mwptoolkit.data.dataloader.multi_equation_dataloader.MultiEquationDataLoader*, *mwptoolkit.data.dataloader.abstract_dataloader.AbstractDataLoader*]]

Create dataloader according to config

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

Returns Dataloader module

mwptoolkit.data.utils.**get_dataset_module**(*config:* mwptoolkit.config.configuration.Config) → Type[Union[*mwptoolkit.data.dataset.dataset_multiencdec.DatasetMultiEncDec*, *mwptoolkit.data.dataset.dataset_ept.DatasetEPT*, *mwptoolkit.data.dataset.dataset_hms.DatasetHMS*, mwptoolkit.data.dataset.dataset_gpt2.DatasetGPT2, *mwptoolkit.data.dataset.pretrain_dataset.PretrainDataset*, *mwptoolkit.data.dataset.single_equation_dataset.SingleEquationDataset*, *mwptoolkit.data.dataset.multi_equation_dataset.MultiEquationDataset*, *mwptoolkit.data.dataset.abstract_dataset.AbstractDataset*]]

return a dataset module according to config

Parameters **config** – An instance object of Config, used to record parameter information.

Returns dataset module

CHAPTER

THREE

MWPTOOLKIT.EVALUATE.EVALUATOR

class mwptoolkit.evaluate.evaluator.**AbstractEvaluator**(*config*)

Bases: object

abstract evaluator

result()

result_multi()

class mwptoolkit.evaluate.evaluator.**InfixEvaluator**(*config*)

Bases: *mwptoolkit.evaluate.evaluator.AbstractEvaluator*

evaluator for infix equation sequnence.

_compute_expression_by_postfix_multi(*expression*)

return solves and unknown number list

result(*test_exp*, *tar_exp*)

evaluate single equation.

Parameters

- **test_exp** (*list*) – list of test expression.
- **tar_exp** (*list*) – list of target expression.

Returns

(tuple(bool,bool,list,list))

val_ac (bool): the correctness of test expression answer compared to target expression answer.

equ_ac (bool): the correctness of test expression compared to target expression.

test_exp (list): list of test expression.

tar_exp (list): iist of target expression.

result_multi(*test_exp*, *tar_exp*)

evaluate multiple euqations.

Parameters

- **test_exp** (*list*) – list of test expression.
- **tar_exp** (*list*) – list of target expression.

Returns

(tuple(bool,bool,list,list))

val_ac (bool): the correctness of test expression answer compared to target expression answer.

equ_ac (bool): the correctness of test expression compared to target expression.

test_exp (list): list of test expression.

tar_exp (list): list of target expression.

class mwptoolkit.evaluate.evaluator.**MultiEncDecEvaluator**(*config*)

Bases: *mwptoolkit.evaluate.evaluator.PostfixEvaluator*, *mwptoolkit.evaluate.evaluator.PrefixEvaluator*

evaluator for deep-learning model MultiE&D.

postfix_result(*test_exp*, *tar_exp*)

evaluate single postfix equation.

Parameters

- **test_exp** (*list*) – list of test expression.
- **tar_exp** (*list*) – list of target expression.

Returns

(tuple(bool,bool,list,list))

val_ac (bool): the correctness of test expression answer compared to target expression answer.

equ_ac (bool): the correctness of test expression compared to target expression.

test_exp (list): list of test expression.

tar_exp (list): list of target expression.

postfix_result_multi(*test_exp*, *tar_exp*)

evaluate multiple postfix euqations.

Parameters

- **test_exp** (*list*) – list of test expression.
- **tar_exp** (*list*) – list of target expression.

Returns

(tuple(bool,bool,list,list))

val_ac (bool): the correctness of test expression answer compared to target expression answer.

equ_ac (bool): the correctness of test expression compared to target expression.

test_exp (list): list of test expression.

tar_exp (list): list of target expression.

prefix_result(*test_exp*, *tar_exp*)

evaluate single prefix equation.

Parameters

- **test_exp** (*list*) – list of test expression.
- **tar_exp** (*list*) – list of target expression.

Returns

(tuple(bool,bool,list,list))

val_ac (bool): the correctness of test expression answer compared to target expression answer.

equ_ac (bool): the correctness of test expression compared to target expression.

test_exp (list): list of test expression.

tar_exp (list): list of target expression.

prefix_result_multi(*test_exp*, *tar_exp*)

evaluate multiple prefix euqations.

Parameters

- **test_exp** (*list*) – list of test expression.
- **tar_exp** (*list*) – list of target expression.

Returns

(tuple(bool,bool,list,list))

val_ac (bool): the correctness of test expression answer compared to target expression answer.

equ_ac (bool): the correctness of test expression compared to target expression.

test_exp (list): list of test expression.

tar_exp (list): list of target expression.

result(*test_exp*, *tar_exp*)

evaluate single equation.

Parameters

- **test_exp** (*list*) – list of test expression.
- **tar_exp** (*list*) – list of target expression.

Returns

(tuple(bool,bool,list,list))

val_ac (bool): the correctness of test expression answer compared to target expression answer.

equ_ac (bool): the correctness of test expression compared to target expression.

test_exp (list): list of test expression.

tar_exp (list): list of target expression.

result_multi(*test_exp*, *tar_exp*)

evaluate multiple euqations.

Parameters

- **test_exp** (*list*) – list of test expression.
- **tar_exp** (*list*) – list of target expression.

Returns

(tuple(bool,bool,list,list))

val_ac (bool): the correctness of test expression answer compared to target expression answer.

equ_ac (bool): the correctness of test expression compared to target expression.

test_exp (list): list of test expression.

tar_exp (list): list of target expression.

class mwptoolkit.evaluate.evaluator.**MultiWayTreeEvaluator**(*config*)

Bases: *mwptoolkit.evaluate.evaluator.AbstractEvaluator*

_compute_expression_by_postfix_multi(*expression*)

return solves and unknown number list

result(*test_exp*, *tar_exp*)

evaluate single equation.

Parameters

- **test_exp** (*list*) – list of test expression.
- **tar_exp** (*list*) – list of target expression.

Returns

(tuple(bool,bool,list,list))

val_ac (bool): the correctness of test expression answer compared to target expression answer.

equ_ac (bool): the correctness of test expression compared to target expression.

test_exp (list): list of test expression.

tar_exp (list): list of target expression.

result_multi(*test_exp*, *tar_exp*)

evaluate multiple euqations.

Parameters

- **test_exp** (*list*) – list of test expression.
- **tar_exp** (*list*) – list of target expression.

Returns

(tuple(bool,bool,list,list))

val_ac (bool): the correctness of test expression answer compared to target expression answer.

equ_ac (bool): the correctness of test expression compared to target expression.

test_exp (list): list of test expression.

tar_exp (list): list of target expression.

class mwptoolkit.evaluate.evaluator.**PostfixEvaluator**(*config*)

Bases: *mwptoolkit.evaluate.evaluator.AbstractEvaluator*

evaluator for postfix equation.

eval_source()

result(*test_exp*, *tar_exp*)

evaluate single equation.

Parameters

- **test_exp** (*list*) – list of test expression.
- **tar_exp** (*list*) – list of target expression.

Returns

(tuple(bool,bool,list,list))

val_ac (bool): the correctness of test expression answer compared to target expression answer.

equ_ac (bool): the correctness of test expression compared to target expression.

test_exp (list): list of test expression.

tar_exp (list): list of target expression.

result_multi(*test_exp*, *tar_exp*)

evaluate multiple euqations.

Parameters

- **test_exp** (*list*) – list of test expression.
- **tar_exp** (*list*) – list of target expression.

Returns

(tuple(bool,bool,list,list))

val_ac (bool): the correctness of test expression answer compared to target expression answer.

equ_ac (bool): the correctness of test expression compared to target expression.

test_exp (list): list of test expression.

tar_exp (list): list of target expression.

class mwptoolkit.evaluate.evaluator.**PrefixEvaluator**(*config*)

Bases: *mwptoolkit.evaluate.evaluator.AbstractEvaluator*

evaluator for prefix equation.

eval_source(*test_res*, *test_tar*, *num_list*, *num_stack=None*)

result(*test_exp*, *tar_exp*)

evaluate single equation.

Parameters

- **test_exp** (*list*) – list of test expression.
- **tar_exp** (*list*) – list of target expression.

Returns

(tuple(bool,bool,list,list))

val_ac (bool): the correctness of test expression answer compared to target expression answer.

equ_ac (bool): the correctness of test expression compared to target expression.

test_exp (list): list of test expression.

tar_exp (list): list of target expression.

result_multi(*test_exp*, *tar_exp*)

evaluate multiple euqations.

Parameters

- **test_exp** (*list*) – list of test expression.
- **tar_exp** (*list*) – list of target expression.

Returns

(tuple(bool,bool,list,list))

val_ac (bool): the correctness of test expression answer compared to target expression answer.

equ_ac (bool): the correctness of test expression compared to target expression.

test_exp (list): list of test expression.

tar_exp (list): list of target expression.

class mwptoolkit.evaluate.evaluator.**Solver**(*func*, *equations*, *unk_symbol*)

Bases: threading.Thread

time-limited equation-solving machanism based threading.

This constructor should always be called with keyword arguments. Arguments are:

group should be None; reserved for future extension when a ThreadGroup class is implemented.

target is the callable object to be invoked by the run() method. Defaults to None, meaning nothing is called.

name is the thread name. By default, a unique name is constructed of the form "Thread-N" where N is a small decimal number.

args is the argument tuple for the target invocation. Defaults to ().

kwargs is a dictionary of keyword arguments for the target invocation. Defaults to {}.

If a subclass overrides the constructor, it must make sure to invoke the base class constructor (Thread.__init__()) before doing anything else to the thread.

get_result()

return the result

run()

run equation solving process

mwptoolkit.evaluate.evaluator.**get_evaluator**(*config*)

build evaluator

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

Returns Constructed evaluator.

Return type Evaluator

mwptoolkit.evaluate.evaluator.**get_evaluator_module**(*config:* mwptoolkit.config.configuration.Config) → Type[Union[*mwptoolkit.evaluate.evaluator.PrefixEvaluator*, *mwptoolkit.evaluate.evaluator.InfixEvaluator*, *mwptoolkit.evaluate.evaluator.PostfixEvaluator*, *mwptoolkit.evaluate.evaluator.MultiWayTreeEvaluator*, *mwptoolkit.evaluate.evaluator.AbstractEvaluator*, *mwptoolkit.evaluate.evaluator.MultiEncDecEvaluator*]]

return a evaluator module according to config

Parameters **config** – An instance object of Config, used to record parameter information.

Returns evaluator module

CHAPTER

FOUR

MWPTOOLKIT.LOSS

4.1 mwptoolkit.loss.abstract_loss

class mwptoolkit.loss.abstract_loss.**AbstractLoss**(*name*, *criterion*)

Bases: object

backward()

loss backward

eval_batch(*outputs*, *target*)

calculate loss

get_loss()

return loss

reset()

reset loss

4.2 mwptoolkit.loss.binary_cross_entropy_loss

class mwptoolkit.loss.binary_cross_entropy_loss.**BinaryCrossEntropyLoss**

Bases: *mwptoolkit.loss.abstract_loss.AbstractLoss*

add_norm(*norm*)

eval_batch(*outputs*, *target*)

calculate loss

Parameters

- **outputs** (*Tensor*) – output distribution of model.
- **target** (*Tensor*) – target distribution.

get_loss()

return loss

Returns loss (float)

4.3 mwptoolkit.loss.cross_entropy_loss

class mwptoolkit.loss.cross_entropy_loss.**CrossEntropyLoss**(*weight=None*, *mask=None*, *size_average=True*)

Bases: *mwptoolkit.loss.abstract_loss.AbstractLoss*

Parameters

- **weight** (*Tensor, optional*) – a manual rescaling weight given to each class.
- **mask** (*Tensor, optional*) – index of classes to rescale weight

eval_batch(*outputs*, *target*)

calculate loss

Parameters

- **outputs** (*Tensor*) – output distribution of model.
- **target** (*Tensor*) – target classes.

get_loss()

return loss

Returns loss (float)

4.4 mwptoolkit.loss.masked_cross_entropy_loss

class mwptoolkit.loss.masked_cross_entropy_loss.**MaskedCrossEntropyLoss**

Bases: *mwptoolkit.loss.abstract_loss.AbstractLoss*

eval_batch(*outputs*, *target*, *length*)

calculate loss

Parameters

- **outputs** (*Tensor*) – output distribution of model.
- **target** (*Tensor*) – target classes.
- **length** (*Tensor*) – length of target.

get_loss()

return loss

Returns loss (float)

mwptoolkit.loss.masked_cross_entropy_loss.**masked_cross_entropy**(*logits*, *target*, *length*)

Parameters

- **logits** – A Variable containing a FloatTensor of size (batch, max_len, num_classes) which contains the unnormalized probability for each class.
- **target** – A Variable containing a LongTensor of size (batch, max_len) which contains the index of the true class for each corresponding step.
- **length** – A Variable containing a LongTensor of size (batch,) which contains the length of each data in a batch.

Returns An average loss value masked by the length.

Return type loss

mwptoolkit.loss.masked_cross_entropy_loss.**sequence_mask**(*sequence_length*, *max_len=None*)

4.5 mwptoolkit.loss.mse_loss

class mwptoolkit.loss.mse_loss.**MSELoss**

Bases: *mwptoolkit.loss.abstract_loss.AbstractLoss*

eval_batch(*outputs*, *target*)

calculate loss

Parameters

- **outputs** (*Tensor*) – output distribution of model.
- **target** (*Tensor*) – target distribution.

get_loss()

return loss

Returns loss (float)

4.6 mwptoolkit.loss.nll_loss

class mwptoolkit.loss.nll_loss.**NLLLoss**(*weight=None*, *mask=None*, *size_average=True*)

Bases: *mwptoolkit.loss.abstract_loss.AbstractLoss*

Parameters

- **weight** (*Tensor, optional*) – a manual rescaling weight given to each class.
- **mask** (*Tensor, optional*) – index of classes to rescale weight

eval_batch(*outputs*, *target*)

calculate loss

Parameters

- **outputs** (*Tensor*) – output distribution of model.
- **target** (*Tensor*) – target classes.

get_loss()

return loss

Returns loss (float)

4.7 mwptoolkit.loss.smoothed_cross_entropy_loss

class mwptoolkit.loss.smoothed_cross_entropy_loss.**SmoothCrossEntropyLoss**(*weight=None*, *mask=None*, *size_average=True*)

Bases: *mwptoolkit.loss.abstract_loss.AbstractLoss*

Computes cross entropy loss with uniformly smoothed targets.

Cross entropy loss with uniformly smoothed targets.

Parameters

- **smoothing** (*float*) – Label smoothing factor, between 0 and 1 (exclusive; default is 0.1)
- **ignore_index** (*int*) – Index to be ignored. (PAD_ID by default)
- **reduction** (*str*) – Style of reduction to be done. One of 'batchmean'(default), 'none', or 'sum'.

eval_batch(*outputs*, *target*)

calculate loss

Parameters

- **outputs** (*Tensor*) – output distribution of model.
- **target** (*Tensor*) – target classes.

get_loss()

return loss

Returns loss (float)

class mwptoolkit.loss.smoothed_cross_entropy_loss.**SmoothedCrossEntropyLoss**(*smoothing: float = 0.1*, *ignore_index: int = - 1*, *reduction: str = 'batchmean'*)

Bases: torch.nn.modules.module.Module

Computes cross entropy loss with uniformly smoothed targets.

Cross entropy loss with uniformly smoothed targets.

Parameters

- **smoothing** (*float*) – Label smoothing factor, between 0 and 1 (exclusive; default is 0.1)
- **ignore_index** (*int*) – Index to be ignored. (PAD_ID by default)
- **reduction** (*str*) – Style of reduction to be done. One of 'batchmean'(default), 'none', or 'sum'.

forward(*input: torch.Tensor*, *target: torch.LongTensor*) → torch.Tensor

Computes cross entropy loss with uniformly smoothed targets. Since the entropy of smoothed target distribution is always same, we can compute this with KL-divergence.

Parameters

- **input** (*torch.Tensor*) – Log probability for each class. This is a Tensor with shape [B, C]

- **target** (*torch.LongTensor*) – List of target classes. This is a LongTensor with shape [B]

Return type torch.Tensor

Returns Computed loss

training: bool

CHAPTER

FIVE

MWPTOOLKIT.MODEL

5.1 mwptoolkit.model.Seq2Seq

5.1.1 mwptoolkit.model.Seq2Seq.dns

class mwptoolkit.model.Seq2Seq.dns.**DNS**(*config*, *dataset*)

Bases: torch.nn.modules.module.Module

Reference: Wang et al. "Deep Neural Solver for Math Word Problems" in EMNLP 2017.

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

calculate_loss(*batch_data: dict*) → float

Finish forward-propagating, calculating loss and back-propagation.

Parameters **batch_data** – one batch data.

Returns loss value.

batch_data should include keywords 'question', 'ques len' and 'equation'

convert_idx2symbol(*output*, *num_list*)

convert_in_idx_2_out_idx(*output*)

convert_out_idx_2_in_idx(*output*)

decode(*output*)

decoder_forward(*encoder_outputs*, *encoder_hidden*, *decoder_inputs*, *target=None*, *output_all_layers=False*)

encoder_forward(*seq_emb*, *seq_length*, *output_all_layers=False*)

filter_END()

filter_op()

forward(*seq*, *seq_length*, *target=None*, *output_all_layers=False*) → Tuple[torch.Tensor, torch.Tensor, Dict[str, Any]]

Parameters

- **seq** (*torch.Tensor*) – input sequence, shape: [batch_size, seq_length].
- **seq_length** (*torch.Tensor*) – the length of sequence, shape: [batch_size].

- **target** (*torch.Tensor | None*) – target, shape: [batch_size, target_length], default None.
- **output_all_layers** (*bool*) – return output of all layers if output_all_layers is True, default False.

:return : token_logits:[batch_size, output_length, output_size], symbol_outputs:[batch_size,output_length], model_all_outputs. :rtype: tuple(torch.Tensor, torch.Tensor, dict)

init_decoder_inputs(*target*, *device*, *batch_size*)

model_test(*batch_data: dict*) → tuple

Model test.

Parameters **batch_data** – one batch data.

Returns predicted equation, target equation.

batch_data should include keywords 'question', 'ques len', 'equation' and 'num list'.

predict(*batch_data: dict*, *output_all_layers=False*)

predict samples without target. :param dict batch_data: one batch data. :param bool output_all_layers: return all layer outputs of model. :return: token_logits, symbol_outputs, all_layer_outputs

rule1_filter()

if r_t1 in {+, , , /}, then rt will not in {+, , , /,), =}.

rule2_filter()

if r_t-1 is a number, then r_t will not be a number and not in {(, =)}.

rule3_filter()

if rt1 is '=', then rt will not in {+, , , /, =,)}.

rule4_filter()

if r_t-1 is '(' , then r_t will not in {(,), +, -, *, /, =}).

rule5_filter()

if r_t1 is ')', then r_t will not be a number and not in {(,)};

rule_filter_(*symbols*, *token_logit*)

Parameters

- **symbols** (*torch.Tensor*) – [batch_size]
- **token_logit** (*torch.Tensor*) – [batch_size, symbol_size]

Returns [batch_size]

Return type symbols of next step (torch.Tensor)

training: bool

5.1.2 mwptoolkit.model.Seq2Seq.ept

class mwptoolkit.model.Seq2Seq.ept.**EPT**(*config*, *dataset*)

Bases: torch.nn.modules.module.Module

Reference: Kim et al. "Point to the Expression: Solving Algebraic Word Problems using the Expression-Pointer Transformer Model" in EMNLP 2020.

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

calculate_loss(*batch_data: dict*) → float

Finish forward-propagating, calculating loss and back-propagation.

Parameters **batch_data** – one batch data.

Returns loss value.

batch_data should include keywords 'question', 'ques len', 'equation','ques mask', 'num pos', 'num size' and 'max numbers'.

convert_idx2symbol(*output*, *num_list*)

batch_size=1

decode(*output*)

decoder_forward(*encoder_output*, *text_num*, *text_numpad*, *src_mask*, *target=None*, *output_all_layers=False*)

encoder_forward(*src*, *src_mask*, *output_all_layers=False*)

forward(*src*, *src_mask*, *num_pos*, *num_size*, *target=None*, *output_all_layers=False*)

Parameters

- **src** (*torch.Tensor*) – input sequence.
- **src_mask** (*list*) – mask of input sequence.
- **num_pos** (*list*) – number position of input sequence.
- **num_size** (*list*) – number of numbers of input sequence.
- **target** (*torch.Tensor*) – target, default None.
- **output_all_layers** (*bool*) – return output of all layers if output_all_layers is True, default False.

Returns token_logits:[batch_size, output_length, output_size], symbol_outputs:[batch_size,output_length], model_all_outputs.

gather_vectors(*hidden: torch.Tensor*, *mask: torch.Tensor*, *max_len: int = 1*)

Gather hidden states of indicated positions.

Parameters

- **hidden** (*torch.Tensor*) – Float Tensor of hidden states. Shape [B, S, H], where B = batch size, S = length of sequence, and H = hidden dimension
- **mask** (*torch.Tensor*) – Long Tensor which indicates number indices that we're interested in. Shape [B, S].
- **max_len** (*int*) – Expected maximum length of vectors per batch. 1 by default.

Return type Tuple[torch.Tensor, torch.Tensor]

Returns

Tuple of Tensors: - [0]: Float Tensor of indicated hidden states.

Shape [B, N, H], where N = max(number of interested positions, max_len)

- **[1]: Bool Tensor of padded positions.** Shape [B, N].

model_test(*batch_data: dict*) → tuple

Model test.

Parameters **batch_data** – one batch data.

Returns predicted equation, target equation.

batch_data should include keywords 'question', 'equation','ques mask', 'num pos', 'num size'.

out_expression_expr(*item*, *num_list*)

out_expression_op(*item*, *num_list*)

predict(*batch_data: dict*, *output_all_layers=False*)

predict samples without target.

Parameters

- **batch_data** (*dict*) – one batch data.
- **output_all_layers** (*bool*) – return all layer outputs of model.

Returns token_logits, symbol_outputs, all_layer_outputs

shift_target(*target: torch.Tensor*, *fill_value=- 1*) → torch.Tensor

Shift matrix to build generation targets.

Parameters

- **target** (*torch.Tensor*) – Target tensor to build generation targets. Shape [B, T]
- **fill_value** – Value to be filled at the padded positions.

Return type torch.Tensor

Returns Tensor with shape [B, T], where (i, j)-entries are (i, j+1) entry of target tensor.

training: bool

mwptoolkit.model.Seq2Seq.ept.**Submodule_types**(*decoder_type*)

5.1.3 mwptoolkit.model.Seq2Seq.groupatt

class mwptoolkit.model.Seq2Seq.groupatt.**GroupATT**(*config*, *dataset*)

Bases: torch.nn.modules.module.Module

Reference: Li et al. "Modeling Intra-Relation in Math Word Problems with Different Functional Multi-Head Attentions" in ACL 2019.

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

calculate_loss(*batch_data: dict*) → float

Finish forward-propagating, calculating loss and back-propagation.

Parameters **batch_data** – one batch data. batch_data should include keywords 'question', 'ques len', 'equation'.

Returns loss value.

convert_idx2symbol(*output*, *num_list*)

convert_in_idx_2_out_idx(*output*)

convert_out_idx_2_in_idx(*output*)

decode(*output*)

decoder_forward(*encoder_outputs*, *encoder_hidden*, *decoder_inputs*, *target=None*, *output_all_layers=False*)

encoder_forward(*seq_emb*, *seq*, *seq_length*, *output_all_layers=False*)

forward(*seq*, *seq_length*, *target=None*, *output_all_layers=False*) → Tuple[torch.Tensor, torch.Tensor, Dict[str, Any]]

Parameters

- **seq** (*torch.Tensor*) – input sequence, shape: [batch_size, seq_length].
- **seq_length** (*torch.Tensor*) – the length of sequence, shape: [batch_size].
- **target** (*torch.Tensor | None*) – target, shape: [batch_size, target_length], default None.
- **output_all_layers** (*bool*) – return output of all layers if output_all_layers is True, default False.

:return : token_logits:[batch_size, output_length, output_size], symbol_outputs:[batch_size,output_length], model_all_outputs. :rtype: tuple(torch.Tensor, torch.Tensor, dict)

init_decoder_inputs(*target*, *device*, *batch_size*)

model_test(*batch_data: dict*) → tuple

Model test.

Parameters **batch_data** – one batch data.

Returns predicted equation, target equation.

batch_data should include keywords 'question', 'ques len', 'equation' and 'num list'.

predict(*batch_data: dict*, *output_all_layers=False*)

predict samples without target.

Parameters

- **batch_data** (*dict*) – one batch data.
- **output_all_layers** (*bool*) – return all layer outputs of model.

Returns token_logits, symbol_outputs, all_layer_outputs

process_gap_encoder_decoder(*encoder_hidden*)

training: bool

5.1.4 mwptoolkit.model.Seq2Seq.mathen

class mwptoolkit.model.Seq2Seq.mathen.**MathEN**(*config*, *dataset*)

Bases: torch.nn.modules.module.Module

Reference: Wang et al. "Translating a Math Word Problem to a Expression Tree" in EMNLP 2018.

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

calculate_loss(*batch_data: dict*) → float

Finish forward-propagating, calculating loss and back-propagation.

Parameters **batch_data** – one batch data.

Returns loss value.

batch_data should include keywords 'question', 'ques len', 'equation', 'ques mask'.

convert_idx2symbol(*output*, *num_list*)

convert_in_idx_2_out_idx(*output*)

convert_out_idx_2_in_idx(*output*)

decode(*output*)

decoder_forward(*encoder_outputs*, *encoder_hidden*, *decoder_inputs*, *target=None*, *output_all_layers=False*)

encoder_forward(*seq_emb*, *seq_length*, *output_all_layers=False*)

forward(*seq*, *seq_length*, *seq_mask=None*, *target=None*, *output_all_layers=False*) → Tuple[torch.Tensor, torch.Tensor, Dict[str, Any]]

Parameters

- **seq** (*torch.Tensor*) – input sequence, shape: [batch_size, seq_length].
- **seq_length** (*torch.Tensor*) – the length of sequence, shape: [batch_size].
- **seq_mask** (*torch.Tensor | None*) – mask of sequence, shape: [batch_size, seq_length], default None.
- **target** (*torch.Tensor | None*) – target, shape: [batch_size, target_length], default None.
- **output_all_layers** (*bool*) – return output of all layers if output_all_layers is True, default False.

:return : token_logits:[batch_size, output_length, output_size], symbol_outputs:[batch_size,output_length], model_all_outputs. :rtype: tuple(torch.Tensor, torch.Tensor, dict)

init_decoder_inputs(*target*, *device*, *batch_size*)

model_test(*batch_data: dict*) → tuple

Model test.

Parameters **batch_data** – one batch data.

Returns predicted equation, target equation.

batch_data should include keywords 'question', 'ques len', 'equation', 'num list', 'ques mask'.

predict(*batch_data: dict*, *output_all_layers=False*)

predict samples without target.

Parameters

- **batch_data** (*dict*) – one batch data.
- **output_all_layers** (*bool*) – return all layer outputs of model.

Returns token_logits, symbol_outputs, all_layer_outputs

training: bool

5.1.5 mwptoolkit.model.Seq2Seq.rnnencdec

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

Bases: torch.nn.modules.module.Module

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

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

calculate_loss(*batch_data: dict*) → float

Finish forward-propagating, calculating loss and back-propagation.

Parameters **batch_data** – one batch data.

Returns loss value.

batch_data should include keywords 'question', 'ques len', 'equation'.

convert_idx2symbol(*output*, *num_list*)

convert_in_idx_2_out_idx(*output*)

convert_out_idx_2_in_idx(*output*)

decode(*output*)

decoder_forward(*encoder_outputs*, *encoder_hidden*, *decoder_inputs*, *target=None*, *output_all_layers=False*)

encoder_forward(*seq_emb*, *seq_length*, *output_all_layers=False*)

forward(*seq*, *seq_length*, *target=None*, *output_all_layers=False*) → Tuple[torch.Tensor, torch.Tensor, Dict[str, Any]]

Parameters

- **seq** (*torch.Tensor*) – input sequence, shape: [batch_size, seq_length].
- **seq_length** (*torch.Tensor*) – the length of sequence, shape: [batch_size].
- **target** (*torch.Tensor | None*) – target, shape: [batch_size, target_length], default None.
- **output_all_layers** (*bool*) – return output of all layers if output_all_layers is True, default False.

:return : token_logits:[batch_size, output_length, output_size], symbol_outputs:[batch_size,output_length], model_all_outputs. :rtype: tuple(torch.Tensor, torch.Tensor, dict)

init_decoder_inputs(*target*, *device*, *batch_size*)

model_test(*batch_data: dict*) → tuple

Model test.

Parameters **batch_data** – one batch data.

Returns predicted equation, target equation.

batch_data should include keywords 'question', 'ques len', 'equation' and 'num list'.

predict(*batch_data: dict*, *output_all_layers=False*)

predict samples without target.

Parameters

- **batch_data** (*dict*) – one batch data.
- **output_all_layers** (*bool*) – return all layer outputs of model.

Returns token_logits, symbol_outputs, all_layer_outputs

training: bool

5.1.6 mwptoolkit.model.Seq2Seq.rnnvae

class mwptoolkit.model.Seq2Seq.rnnvae.**RNNVAE**(*config*, *dataset*)

Bases: torch.nn.modules.module.Module

Reference: Zhang et al. "Variational Neural Machine Translation".

We apply translation machine based rnnvae to math word problem task.

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

calculate_loss(*batch_data: dict*) → float

Finish forward-propagating, calculating loss and back-propagation.

Parameters **batch_data** – one batch data.

Returns loss value.

batch_data should include keywords 'question', 'ques len', 'equation'.

convert_idx2symbol(*output*, *num_list*)

convert_in_idx_2_out_idx(*output*)

convert_out_idx_2_in_idx(*output*)

decode(*output*)

decoder_forward(*encoder_outputs*, *encoder_hidden*, *decoder_inputs*, *z*, *target=None*, *output_all_layers=False*)

encoder_forward(*seq_emb*, *seq_length*, *output_all_layers=False*)

forward(*seq*, *seq_length*, *target=None*, *output_all_layers=False*) → Tuple[torch.Tensor, torch.Tensor, Dict[str, Any]]

Parameters

- **seq** (*torch.Tensor*) – input sequence, shape: [batch_size, seq_length].

- **seq_length** (*torch.Tensor*) – the length of sequence, shape: [batch_size].
- **target** (*torch.Tensor | None*) – target, shape: [batch_size, target_length], default None.
- **output_all_layers** (*bool*) – return output of all layers if output_all_layers is True, default False.

:return : token_logits:[batch_size, output_length, output_size], symbol_outputs:[batch_size,output_length], model_all_outputs. :rtype: tuple(torch.Tensor, torch.Tensor, dict)

init_decoder_inputs(*target*, *device*, *batch_size*)

model_test(*batch_data: dict*) → tuple

Model test.

Parameters **batch_data** – one batch data.

Returns predicted equation, target equation.

batch_data should include keywords 'question', 'ques len', 'equation' and 'num list'.

predict(*batch_data: dict*, *output_all_layers=False*)

predict samples without target.

Parameters

- **batch_data** (*dict*) – one batch data.
- **output_all_layers** (*bool*) – return all layer outputs of model.

Returns token_logits, symbol_outputs, all_layer_outputs

training: bool

5.1.7 mwptoolkit.model.Seq2Seq.saligned

class mwptoolkit.model.Seq2Seq.saligned.**Saligned**(*config*, *dataset*)

Bases: torch.nn.modules.module.Module

Reference: Chiang et al. "Semantically-Aligned Equation Generation for Solving and Reasoning Math Word Problems".

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

calculate_loss(*batch_data: dict*) → float

Finish forward-propagating, calculating loss and back-propagation.

Parameters **batch_data** – one batch data.

Returns loss value.

batch_data should include keywords 'question', 'ques len', 'equation', 'equ len', 'num pos', 'num list', 'num size'.

convert_idx2symbol(*output*, *num_list*)

convert_mask_num(*batch_output*, *num_list*)

decoder_forward(*encoder_outputs*, *encoder_hidden*, *inputs_length*, *operands*, *stacks*, *number_emb*, *target=None*, *target_length=None*, *output_all_layers=False*)

encoder_forward(*seq_emb*, *seq_length*, *constant_indices*, *output_all_layers=False*)

forward(*seq*, *seq_length*, *number_list*, *number_position*, *number_size*, *target=None*, *target_length=None*, *output_all_layers=False*) → Tuple[Tuple[torch.Tensor, torch.Tensor], torch.Tensor, Dict[str, Any]]

Parameters

- **seq** (*torch.Tensor*) –
- **seq_length** (*torch.Tensor*) –
- **number_list** (*list*) –
- **number_position** (*list*) –
- **number_size** (*list*) –
- **target** (*torch.Tensor | None*) –
- **target_length** (*torch.Tensor | None*) –
- **output_all_layers** (*bool*) –

Returns token_logits:[batch_size, output_length, output_size], symbol_outputs:[batch_size,output_length], model_all_outputs.

Return type tuple(torch.Tensor, torch.Tensor, dict)

model_test(*batch_data: dict*) → tuple

Model test.

Parameters **batch_data** – one batch data.

Returns predicted equation, target equation.

batch_data should include keywords 'question', 'ques len', 'equation', 'equ len', 'num pos', 'num list', 'num size'.

predict(*batch_data: dict*, *output_all_layers=False*)

predict samples without target.

Parameters

- **batch_data** (*dict*) – one batch data.
- **output_all_layers** (*bool*) – return all layer outputs of model.

Returns token_logits, symbol_outputs, all_layer_outputs

training: bool

5.1.8 mwptoolkit.model.Seq2Seq.transformer

class mwptoolkit.model.Seq2Seq.transformer.**Transformer**(*config*, *dataset*)

Bases: torch.nn.modules.module.Module

Reference: Vaswani et al. "Attention Is All You Need".

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

calculate_loss(*batch_data: dict*) → float

Finish forward-propagating, calculating loss and back-propagation.

Parameters **batch_data** – one batch data.

Returns loss value.

batch_data should include keywords 'question', 'equation'.

convert_idx2symbol(*output*, *num_list*)

convert_in_idx_2_out_idx(*output*)

convert_out_idx_2_in_idx(*output*)

decode(*output*)

decoder_forward(*encoder_outputs*, *seq_mask*, *target=None*, *output_all_layers=False*)

encoder_forward(*seq_emb*, *seq_mask*, *output_all_layers=False*)

forward(*src*, *target=None*, *output_all_layers=False*) → Tuple[torch.Tensor, torch.Tensor, Dict[str, Any]]

Parameters

- **src** (*torch.Tensor*) – input sequence, shape: [batch_size, seq_length].
- **target** (*torch.Tensor|None*) – target, shape: [batch_size, target_length], default None.
- **output_all_layers** (*bool*) – default False, return output of all layers if output_all_layers is True.

Returns token_logits, symbol_outputs, model_all_outputs.

:rtype tuple(torch.Tensor, torch.Tensor, dict)

init_decoder_inputs(*target*, *device*, *batch_size*)

model_test(*batch_data: dict*) → tuple

Model test.

Parameters **batch_data** – one batch data.

Returns predicted equation, target equation.

batch_data should include keywords 'question', 'equation' and 'num list'.

predict(*batch_data: dict*, *output_all_layers=False*)

predict samples without target.

Parameters

- **batch_data** (*dict*) – one batch data.
- **output_all_layers** (*bool*) – return all layer outputs of model.

Returns token_logits, symbol_outputs, all_layer_outputs

training: bool

5.2 mwptoolkit.model.Seq2Tree

5.2.1 mwptoolkit.model.Seq2Tree.berttd

class mwptoolkit.model.Seq2Tree.berttd.**BertTD**(*config*, *dataset*)

Bases: torch.nn.modules.module.Module

Reference: Li et al. Seeking Patterns, Not just Memorizing Procedures: Contrastive Learning for Solving Math Word Problems

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

calculate_loss(*batch_data: dict*) → float

Finish forward-propagating, calculating loss and back-propagation.

Parameters **batch_data** – one batch data.

Returns loss value.

batch_data should include keywords 'question', 'ques len', 'equation', 'equ len', 'num stack', 'num size', 'num pos'

convert_idx2symbol(*output*, *num_list*, *num_stack*)

batch_size=1

decoder_forward(*encoder_outputs*, *problem_output*, *all_nums_encoder_outputs*, *nums_stack*, *seq_mask*, *num_mask*, *target=None*, *output_all_layers=False*)

encoder_forward(*seq*, *seq_mask*, *output_all_layers=False*)

forward(*seq*, *seq_length*, *nums_stack*, *num_size*, *num_pos*, *target=None*, *output_all_layers=False*) → Tuple[torch.Tensor, torch.Tensor, Dict[str, Any]]

Parameters

- **seq** (*torch.Tensor*) – input sequence, shape: [batch_size, seq_length].
- **seq_length** (*torch.Tensor*) – the length of sequence, shape: [batch_size].
- **nums_stack** (*list*) – different positions of the same number, length:[batch_size]
- **num_size** (*list*) – number of numbers of input sequence, length:[batch_size].
- **num_pos** (*list*) – number positions of input sequence, length:[batch_size].
- **target** (*torch.Tensor | None*) – target, shape: [batch_size, target_length], default None.
- **output_all_layers** (*bool*) – return output of all layers if output_all_layers is True, default False.

:return : token_logits:[batch_size, output_length, output_size], symbol_outputs:[batch_size,output_length], model_all_outputs. :rtype: tuple(torch.Tensor, torch.Tensor, dict)

generate_tree_input(*target*, *decoder_output*, *nums_stack_batch*, *num_start*, *unk*)

get_all_number_encoder_outputs(*encoder_outputs*, *num_pos*, *batch_size*, *num_size*, *hidden_size*)

model_test(*batch_data: dict*) → tuple

Model test.

Parameters **batch_data** – one batch data.

Returns predicted equation, target equation.

batch_data should include keywords 'question', 'ques len', 'equation', 'num stack', 'num pos', 'num list'

predict(*batch_data: dict*, *output_all_layers=False*)

predict samples without target.

Parameters

- **batch_data** (*dict*) – one batch data.
- **output_all_layers** (*bool*) – return all layer outputs of model.

Returns token_logits, symbol_outputs, all_layer_outputs

training: bool

5.2.2 mwptoolkit.model.Seq2Tree.gts

class mwptoolkit.model.Seq2Tree.gts.**GTS**(*config*, *dataset*)

Bases: torch.nn.modules.module.Module

Reference: Xie et al. "A Goal-Driven Tree-Structured Neural Model for Math Word Problems" in IJCAI 2019.

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

calculate_loss(*batch_data: dict*) → float

Finish forward-propagating, calculating loss and back-propagation.

Parameters **batch_data** – one batch data.

Returns loss value.

batch_data should include keywords 'question', 'ques len', 'equation', 'equ len', 'num stack', 'num size', 'num pos'

convert_idx2symbol(*output*, *num_list*, *num_stack*)

batch_size=1

decoder_forward(*encoder_outputs*, *problem_output*, *all_nums_encoder_outputs*, *nums_stack*, *seq_mask*, *num_mask*, *target=None*, *output_all_layers=False*)

encoder_forward(*seq_emb*, *seq_length*, *output_all_layers=False*)

forward(*seq*, *seq_length*, *nums_stack*, *num_size*, *num_pos*, *target=None*, *output_all_layers=False*) → Tuple[torch.Tensor, torch.Tensor, Dict[str, Any]]

Parameters

- **seq** (*torch.Tensor*) – input sequence, shape: [batch_size, seq_length].
- **seq_length** (*torch.Tensor*) – the length of sequence, shape: [batch_size].
- **nums_stack** (*list*) – different positions of the same number, length:[batch_size]
- **num_size** (*list*) – number of numbers of input sequence, length:[batch_size].
- **num_pos** (*list*) – number positions of input sequence, length:[batch_size].

- **target** (*torch.Tensor | None*) – target, shape: [batch_size, target_length], default None.
- **output_all_layers** (*bool*) – return output of all layers if output_all_layers is True, default False.

:return : token_logits:[batch_size, output_length, output_size], symbol_outputs:[batch_size,output_length], model_all_outputs. :rtype: tuple(torch.Tensor, torch.Tensor, dict)

generate_tree_input(*target*, *decoder_output*, *nums_stack_batch*, *num_start*, *unk*)

get_all_number_encoder_outputs(*encoder_outputs*, *num_pos*, *batch_size*, *num_size*, *hidden_size*)

model_test(*batch_data: dict*) → tuple

Model test.

Parameters **batch_data** – one batch data.

Returns predicted equation, target equation.

batch_data should include keywords 'question', 'ques len', 'equation', 'num stack', 'num pos', 'num list','num size'

predict(*batch_data: dict*, *output_all_layers=False*)

predict samples without target.

Parameters

- **batch_data** (*dict*) – one batch data.
- **output_all_layers** (*bool*) – return all layer outputs of model.

Returns token_logits, symbol_outputs, all_layer_outputs

training: bool

5.2.3 mwptoolkit.model.Seq2Tree.mwpbert

class mwptoolkit.model.Seq2Tree.mwpbert.**MWPBert**(*config*, *dataset*)

Bases: torch.nn.modules.module.Module

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

calculate_loss(*batch_data: dict*) → float

Finish forward-propagating, calculating loss and back-propagation.

Parameters **batch_data** – one batch data.

Returns loss value.

batch_data should include keywords 'question', 'ques len', 'equation', 'equ len', 'num stack', 'num size', 'num pos'

convert_idx2symbol(*output*, *num_list*, *num_stack*)

batch_size=1

decoder_forward(*encoder_outputs*, *problem_output*, *all_nums_encoder_outputs*, *nums_stack*, *seq_mask*, *num_mask*, *target=None*, *output_all_layers=False*)

encoder_forward(*seq*, *seq_mask*, *seq_length*, *output_all_layers=False*)

forward(*seq*, *seq_length*, *nums_stack*, *num_size*, *num_pos*, *target=None*, *output_all_layers=False*) → Tuple[torch.Tensor, torch.Tensor, Dict[str, Any]]

Parameters

- **seq** (*torch.Tensor*) – input sequence, shape: [batch_size, seq_length].
- **seq_length** (*torch.Tensor*) – the length of sequence, shape: [batch_size].
- **nums_stack** (*list*) – different positions of the same number, length:[batch_size]
- **num_size** (*list*) – number of numbers of input sequence, length:[batch_size].
- **num_pos** (*list*) – number positions of input sequence, length:[batch_size].
- **target** (*torch.Tensor | None*) – target, shape: [batch_size, target_length], default None.
- **output_all_layers** (*bool*) – return output of all layers if output_all_layers is True, default False.

:return : token_logits:[batch_size, output_length, output_size], symbol_outputs:[batch_size,output_length], model_all_outputs. :rtype: tuple(torch.Tensor, torch.Tensor, dict)

generate_tree_input(*target*, *decoder_output*, *nums_stack_batch*, *num_start*, *unk*)

get_all_number_encoder_outputs(*encoder_outputs*, *num_pos*, *batch_size*, *num_size*, *hidden_size*)

model_test(*batch_data: dict*) → tuple

Model test.

Parameters **batch_data** – one batch data.

Returns predicted equation, target equation.

batch_data should include keywords 'question', 'ques len', 'equation', 'num stack', 'num pos', 'num list','num size'

predict(*batch_data: dict*, *output_all_layers=False*)

predict samples without target.

Parameters

- **batch_data** (*dict*) – one batch data.
- **output_all_layers** (*bool*) – return all layer outputs of model.

Returns token_logits, symbol_outputs, all_layer_outputs

training: bool

5.2.4 mwptoolkit.model.Seq2Tree.sausolver

class mwptoolkit.model.Seq2Tree.sausolver.**SAUSolver**(*config*, *dataset*)

Bases: torch.nn.modules.module.Module

Reference: Qin et al. "Semantically-Aligned Universal Tree-Structured Solver for Math Word Problems" in EMNLP 2020.

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

calculate_loss(*batch_data: dict*) → float

Finish forward-propagating, calculating loss and back-propagation.

Parameters batch_data – one batch data.

Returns loss value.

batch_data should include keywords 'question', 'ques len', 'equation', 'equ len', 'num stack', 'num size', 'num pos'

convert_idx2symbol(*output*, *num_list*, *num_stack*)

batch_size=1

decoder_forward(*encoder_outputs*, *problem_output*, *all_nums_encoder_outputs*, *nums_stack*, *seq_mask*, *num_mask*, *target=None*, *output_all_layers=False*)

encoder_forward(*seq_emb*, *seq_length*, *output_all_layers=False*)

evaluate_tree(*input_batch*, *input_length*, *generate_nums*, *num_pos*, *num_start*, *beam_size=5*, *max_length=30*)

forward(*seq*, *seq_length*, *nums_stack*, *num_size*, *num_pos*, *target=None*, *output_all_layers=False*) → Tuple[torch.Tensor, torch.Tensor, Dict[str, Any]]

Parameters

- **seq** (*torch.Tensor*) – input sequence, shape: [batch_size, seq_length].
- **seq_length** (*torch.Tensor*) – the length of sequence, shape: [batch_size].
- **nums_stack** (*list*) – different positions of the same number, length:[batch_size]
- **num_size** (*list*) – number of numbers of input sequence, length:[batch_size].
- **num_pos** (*list*) – number positions of input sequence, length:[batch_size].
- **target** (*torch.Tensor | None*) – target, shape: [batch_size, target_length], default None.
- **output_all_layers** (*bool*) – return output of all layers if output_all_layers is True, default False.

:return : token_logits:[batch_size, output_length, output_size], symbol_outputs:[batch_size,output_length], model_all_outputs. :rtype: tuple(torch.Tensor, torch.Tensor, dict)

generate_tree_input(*target*, *decoder_output*, *nums_stack_batch*, *num_start*, *unk*)

get_all_number_encoder_outputs(*encoder_outputs*, *num_pos*, *batch_size*, *num_size*, *hidden_size*)

model_test(*batch_data: dict*) → tuple

Model test.

Parameters batch_data – one batch data.

Returns predicted equation, target equation.

batch_data should include keywords 'question', 'ques len', 'equation', 'num stack', 'num pos', 'num list'

mse_loss(*outputs*, *targets*, *mask=None*)

predict(*batch_data: dict*, *output_all_layers=False*)

predict samples without target.

Parameters

- **batch_data** (*dict*) – one batch data.
- **output_all_layers** (*bool*) – return all layer outputs of model.

Returns token_logits, symbol_outputs, all_layer_outputs

train_tree(*input_batch*, *input_length*, *target_batch*, *target_length*, *nums_stack_batch*, *num_size_batch*, *generate_nums*, *num_pos*, *unk*, *num_start*, *english=False*, *var_nums=[]*, *batch_first=False*)

training: bool

5.2.5 mwptoolkit.model.Seq2Tree.treelstm

class mwptoolkit.model.Seq2Tree.treelstm.**TreeLSTM**(*config*, *dataset*)

Bases: torch.nn.modules.module.Module

Reference: Liu et al. "Tree-structured Decoding for Solving Math Word Problems" in EMNLP | IJCNLP 2019.

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

calculate_loss(*batch_data: dict*) → float

Finish forward-propagating, calculating loss and back-propagation.

Parameters **batch_data** – one batch data.

Returns loss value.

batch_data should include keywords 'question', 'ques len', 'equation', 'equ len', 'num stack', 'num size', 'num pos'

convert_idx2symbol(*output*, *num_list*, *num_stack*)

batch_size=1

copy_list(*l*)

decoder_forward(*encoder_outputs*, *initial_hidden*, *problem_output*, *all_nums_encoder_outputs*, *seq_mask*, *num_mask*, *nums_stack*, *target=None*, *output_all_layers=False*)

encoder_forward(*seq_emb*, *output_all_layers=False*)

forward(*seq*, *seq_length*, *nums_stack*, *num_size*, *num_pos*, *target=None*, *output_all_layers=False*) → Tuple[torch.Tensor, torch.Tensor, Dict[str, Any]]

Parameters

- **seq** (*torch.Tensor*) – input sequence, shape: [batch_size, seq_length].
- **seq_length** (*torch.Tensor*) – the length of sequence, shape: [batch_size].
- **nums_stack** (*list*) – different positions of the same number, length:[batch_size]
- **num_size** (*list*) – number of numbers of input sequence, length:[batch_size].
- **num_pos** (*list*) – number positions of input sequence, length:[batch_size].
- **target** (*torch.Tensor | None*) – target, shape: [batch_size, target_length], default None.

- **output_all_layers** (*bool*) – return output of all layers if output_all_layers is True, default False.

:return : token_logits:[batch_size, output_length, output_size], symbol_outputs:[batch_size,output_length], model_all_outputs. :rtype: tuple(torch.Tensor, torch.Tensor, dict)

generate_tree_input(*target*, *decoder_output*, *nums_stack_batch*, *num_start*, *unk*)

get_all_number_encoder_outputs(*encoder_outputs*, *num_pos*, *num_size*, *hidden_size*)

model_test(*batch_data: dict*) → tuple

Model test.

Parameters **batch_data** – one batch data.

Returns predicted equation, target equation.

batch_data should include keywords 'question', 'ques len', 'equation', 'num stack', 'num pos', num size, 'num list'

predict(*batch_data: dict*, *output_all_layers=False*)

predict samples without target.

Parameters

- **batch_data** (*dict*) – one batch data.
- **output_all_layers** (*bool*) – return all layer outputs of model.

Returns token_logits, symbol_outputs, all_layer_outputs

training: bool

5.2.6 mwptoolkit.model.Seq2Tree.trnn

class mwptoolkit.model.Seq2Tree.trnn.**TRNN**(*config*, *dataset*)

Bases: torch.nn.modules.module.Module

Reference: Wang et al. "Template-Based Math Word Problem Solvers with Recursive Neural Networks" in AAAI 2019.

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

ans_module_calculate_loss(*batch_data*)

Finish forward-propagating, calculating loss and back-propagation of answer module.

Parameters **batch_data** – one batch data.

Returns loss value of answer module.

ans_module_forward(*seq*, *seq_length*, *seq_mask*, *template*, *num_pos*, *equation_target=None*, *output_all_layers=False*)

calculate_loss(*batch_data: dict*) → Tuple[float, float]

Finish forward-propagating, calculating loss and back-propagation.

Parameters **batch_data** – one batch data.

Returns seq2seq module loss, answer module loss.

convert_idx2symbol(*output*, *num_list*)

convert_in_idx_2_temp_idx(*output*)

convert_temp_idx2symbol(*output*)

convert_temp_idx_2_in_idx(*output*)

forward(*seq*, *seq_length*, *seq_mask*, *num_pos*, *template_target=None*, *equation_target=None*, *output_all_layers=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.

---

init_seq2seq_decoder_inputs(*target*, *device*, *batch_size*)

mask2num(*output*, *num_list*)

model_test(*batch_data: dict*) → tuple

Model test.

Parameters **batch_data** – one batch data.

Returns predicted equation, target equation.

batch_data should include keywords 'question', 'ques len', 'equation', 'ques mask', 'num pos', 'num list', 'template'

predict(*batch_data: dict*, *output_all_layers=False*)

predict samples without target.

Parameters

- **batch_data** (*dict*) – one batch data.
- **output_all_layers** (*bool*) – return all layer outputs of model.

Returns token_logits, symbol_outputs, all_layer_outputs

seq2seq_calculate_loss(*batch_data: dict*) → float

Finish forward-propagating, calculating loss and back-propagation of seq2seq module.

Parameters **batch_data** – one batch data.

Returns loss value of seq2seq module.

seq2seq_decoder_forward(*encoder_outputs*, *encoder_hidden*, *decoder_inputs*, *target=None*, *output_all_layers=False*)

seq2seq_encoder_forward(*seq_emb*, *seq_length*, *output_all_layers=False*)

seq2seq_forward(*seq*, *seq_length*, *target=None*, *output_all_layers=False*)

seq2seq_generate_t(*encoder_outputs*, *encoder_hidden*, *decoder_inputs*)

seq2seq_generate_without_t(*encoder_outputs*, *encoder_hidden*, *decoder_input*)

symbol2idx(*symbols*)

symbol to idx equation symbol to equation idx

template2tree(*template*)

training: bool

tree2equation(*tree*)

5.2.7 mwptoolkit.model.Seq2Tree.tsn

class mwptoolkit.model.Seq2Tree.tsn.**TSN**(*config*, *dataset*)

Bases: torch.nn.modules.module.Module

Reference: Zhang et al. "Teacher-Student Networks with Multiple Decoders for Solving Math Word Problem" in IJCAI 2020.

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

build_graph(*seq_length*, *num_list*, *num_pos*, *group_nums*)

convert_idx2symbol(*output*, *num_list*, *num_stack*)

batch_size=1

forward(*seq*, *seq_length*, *nums_stack*, *num_size*, *num_pos*, *target=None*, *output_all_layers=False*)

Parameters

- **seq** –
- **seq_length** –
- **nums_stack** –
- **num_size** –
- **num_pos** –
- **target** –
- **output_all_layers** –

Returns

generate_tree_input(*target*, *decoder_output*, *nums_stack_batch*, *num_start*, *unk*)

get_all_number_encoder_outputs(*encoder_outputs*, *num_pos*, *batch_size*, *num_size*, *hidden_size*)

get_soft_target(*batch_id*)

init_encoder_mask(*batch_size*)

init_soft_target(*batch_data*)

Build soft target

Parameters **batch_data** (*dict*) – one batch data.

model_test(*batch_data*)

predict(*batch_data: dict*, *output_all_layers=False*)

predict samples without target.

Parameters

- **batch_data** (*dict*) – one batch data.
- **output_all_layers** (*bool*) – return all layer outputs of model.

Returns token_logits, symbol_outputs, all_layer_outputs

student_calculate_loss(*batch_data: dict*) → float

Finish forward-propagating, calculating loss and back-propagation of student net.

Parameters **batch_data** – one batch data.

Returns loss value.

batch_data should include keywords 'question', 'ques len', 'equation', 'equ len', 'num stack', 'num size', 'num pos', 'id'

student_net_1_decoder_forward(*encoder_outputs*, *problem_output*, *all_nums_encoder_outputs*, *nums_stack*, *seq_mask*, *num_mask*, *target=None*, *output_all_layers=False*)

student_net_2_decoder_forward(*encoder_outputs*, *problem_output*, *all_nums_encoder_outputs*, *nums_stack*, *seq_mask*, *num_mask*, *target=None*, *output_all_layers=False*)

student_net_decoder_forward(*encoder_outputs*, *problem_output*, *all_nums_encoder_outputs*, *nums_stack*, *seq_mask*, *num_mask*, *target=None*, *output_all_layers=False*)

student_net_encoder_forward(*seq_emb*, *seq_length*, *output_all_layers=False*)

student_net_forward(*seq*, *seq_length*, *nums_stack*, *num_size*, *num_pos*, *target=None*, *output_all_layers=False*) → Tuple[Tuple[torch.Tensor, torch.Tensor], Tuple[torch.Tensor, torch.Tensor], Dict[str, Any]]

Parameters

- **seq** (*torch.Tensor*) – input sequence, shape: [batch_size, seq_length].
- **seq_length** (*torch.Tensor*) – the length of sequence, shape: [batch_size].
- **nums_stack** (*list*) – different positions of the same number, length:[batch_size]
- **num_size** (*list*) – number of numbers of input sequence, length:[batch_size].
- **num_pos** (*list*) – number positions of input sequence, length:[batch_size].
- **target** (*torch.Tensor | None*) – target, shape: [batch_size, target_length], default None.
- **output_all_layers** (*bool*) – return output of all layers if output_all_layers is True, default False.

:return : token_logits:(token_logits_1,token_logits_2), symbol_outputs:(symbol_outputs_1,symbol_outputs_2), model_all_outputs. :rtype: tuple(tuple(torch.Tensor), tuple(torch.Tensor), dict)

student_test(*batch_data: dict*) → Tuple[list, float, list, float, list]

Student net test.

Parameters **batch_data** – one batch data.

Returns predicted equation1, score1, predicted equation2, score2, target equation.

batch_data should include keywords 'question', 'ques len', 'equation', 'num stack', 'num pos', 'num list'

teacher_calculate_loss(*batch_data: dict*) → float

Finish forward-propagating, calculating loss and back-propagation of teacher net.

Parameters **batch_data** – one batch data.

Returns loss value

batch_data should include keywords 'question', 'ques len', 'equation', 'equ len', 'num stack', 'num size', 'num pos'

teacher_net_decoder_forward(*encoder_outputs*, *problem_output*, *all_nums_encoder_outputs*, *nums_stack*, *seq_mask*, *num_mask*, *target=None*, *output_all_layers=False*)

teacher_net_encoder_forward(*seq_emb*, *seq_length*, *output_all_layers=False*)

teacher_net_forward(*seq*, *seq_length*, *nums_stack*, *num_size*, *num_pos*, *target=None*, *output_all_layers=False*) → Tuple[torch.Tensor, torch.Tensor, Dict[str, Any]]

Parameters

- **seq** (*torch.Tensor*) – input sequence, shape: [batch_size, seq_length].
- **seq_length** (*torch.Tensor*) – the length of sequence, shape: [batch_size].
- **nums_stack** (*list*) – different positions of the same number, length:[batch_size]
- **num_size** (*list*) – number of numbers of input sequence, length:[batch_size].
- **num_pos** (*list*) – number positions of input sequence, length:[batch_size].
- **target** (*torch.Tensor | None*) – target, shape: [batch_size, target_length], default None.
- **output_all_layers** (*bool*) – return output of all layers if output_all_layers is True, default False.

:return : token_logits:[batch_size, output_length, output_size], symbol_outputs:[batch_size,output_length], model_all_outputs. :rtype: tuple(torch.Tensor, torch.Tensor, dict)

teacher_test(*batch_data: dict*) → tuple

Teacher net test.

Parameters **batch_data** – one batch data.

Returns predicted equation, target equation.

batch_data should include keywords 'question', 'ques len', 'equation', 'num stack', 'num pos', 'num list'

training: bool

mwptoolkit.model.Seq2Tree.tsn.**cosine_loss**(*logits*, *logits_1*, *length*)

mwptoolkit.model.Seq2Tree.tsn.**cosine_sim**(*logits*, *logits_1*)

mwptoolkit.model.Seq2Tree.tsn.**soft_cross_entropy_loss**(*predict_score*, *label_score*)

mwptoolkit.model.Seq2Tree.tsn.**soft_target_loss**(*logits*, *soft_target*, *length*)

5.3 mwptoolkit.model.Graph2Tree

5.3.1 mwptoolkit.model.Graph2Tree.graph2tree

class mwptoolkit.model.Graph2Tree.graph2tree.**Graph2Tree**(*config*, *dataset*)

Bases: torch.nn.modules.module.Module

Reference: Zhang et al."Graph-to-Tree Learning for Solving Math Word Problems" in ACL 2020.

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

build_graph(*seq_length*, *num_list*, *num_pos*, *group_nums*)

calculate_loss(*batch_data: dict*) → float

Finish forward-propagating, calculating loss and back-propagation.

Parameters **batch_data** – one batch data.

Returns loss value.

batch_data should include keywords 'question', 'ques len', 'equation', 'equ len', 'num stack', 'num size', 'num pos', 'num list', 'group nums'

convert_idx2symbol(*output*, *num_list*, *num_stack*)

batch_size=1

decoder_forward(*encoder_outputs*, *problem_output*, *all_nums_encoder_outputs*, *nums_stack*, *seq_mask*, *num_mask*, *target=None*, *output_all_layers=False*)

encoder_forward(*seq_emb*, *input_length*, *graph*, *output_all_layers=False*)

forward(*seq*, *seq_length*, *nums_stack*, *num_size*, *num_pos*, *num_list*, *group_nums*, *target=None*, *output_all_layers=False*) → Tuple[torch.Tensor, torch.Tensor, Dict[str, Any]]

Parameters

- **seq** (*torch.Tensor*) – input sequence, shape: [batch_size, seq_length].
- **seq_length** (*torch.Tensor*) – the length of sequence, shape: [batch_size].
- **nums_stack** (*list*) – different positions of the same number, length:[batch_size]
- **num_size** (*list*) – number of numbers of input sequence, length:[batch_size].
- **num_pos** (*list*) – number positions of input sequence, length:[batch_size].
- **num_list** (*list*) – numbers of input sequence, length:[batch_size].
- **group_nums** (*list*) – group numbers of input sequence, length:[batch_size].
- **target** (*torch.Tensor | None*) – target, shape: [batch_size, target_length], default None.
- **output_all_layers** (*bool*) – return output of all layers if output_all_layers is True, default False.

:return : token_logits:[batch_size, output_length, output_size], symbol_outputs:[batch_size,output_length], model_all_outputs. :rtype: tuple(torch.Tensor, torch.Tensor, dict)

generate_tree_input(*target*, *decoder_output*, *nums_stack_batch*, *num_start*, *unk*)

get_all_number_encoder_outputs(*encoder_outputs*, *num_pos*, *batch_size*, *num_size*, *hidden_size*)

model_test(*batch_data: dict*) → tuple

Model test.

Parameters **batch_data** – one batch data.

Returns predicted equation, target equation.

batch_data should include keywords 'question', 'ques len', 'equation', 'num stack', 'num pos', 'num list', 'num_size', 'group nums'

predict(*batch_data: dict*, *output_all_layers=False*)

predict samples without target.

Parameters

- **batch_data** (*dict*) – one batch data.
- **output_all_layers** (*bool*) – return all layer outputs of model.

Returns token_logits, symbol_outputs, all_layer_outputs

training: bool

5.3.2 mwptoolkit.model.Graph2Tree.multiencdec

class mwptoolkit.model.Graph2Tree.multiencdec.**MultiEncDec**(*config*, *dataset*)

Bases: torch.nn.modules.module.Module

Reference: Shen et al. "Solving Math Word Problems with Multi-Encoders and Multi-Decoders" in COLING 2020.

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

attn_decoder_forward(*encoder_outputs*, *seq_mask*, *decoder_hidden*, *num_stack*, *target=None*, *output_all_layers=False*)

calculate_loss(*batch_data: dict*) → float

Finish forward-propagating, calculating loss and back-propagation.

Parameters **batch_data** – one batch data.

Returns loss value.

batch_data should include keywords 'input1', 'input2', 'output1', 'output2', 'input1 len', 'parse graph', 'num stack', 'output1 len', 'output2 len', 'num size', 'num pos', 'num order'

convert_idx2symbol1(*output*, *num_list*, *num_stack*)

batch_size=1

convert_idx2symbol2(*output*, *num_list*, *num_stack*)

decoder_forward(*encoder_outputs*, *problem_output*, *attn_decoder_hidden*, *all_nums_encoder_outputs*, *seq_mask*, *num_mask*, *num_stack*, *target1*, *target2*, *output_all_layers*)

encoder_forward(*input1*, *input2*, *input_length*, *parse_graph*, *num_pos*, *num_pos_pad*, *num_order_pad*, *output_all_layers=False*)

forward(*input1*, *input2*, *input_length*, *num_size*, *num_pos*, *num_order*, *parse_graph*, *num_stack*, *target1=None*, *target2=None*, *output_all_layers=False*)

Parameters

- **input1** (*torch.Tensor*) –
- **input2** (*torch.Tensor*) –
- **input_length** (*torch.Tensor*) –
- **num_size** (*list*) –
- **num_pos** (*list*) –
- **num_order** (*list*) –
- **parse_graph** (*torch.Tensor*) –
- **num_stack** (*list*) –
- **target1** (*torch.Tensor | None*) –
- **target2** (*torch.Tensor | None*) –
- **output_all_layers** (*bool*) –

Returns

generate_decoder_input(*target*, *decoder_output*, *nums_stack_batch*, *num_start*, *unk*)

generate_tree_input(*target*, *decoder_output*, *nums_stack_batch*, *num_start*, *unk*)

get_all_number_encoder_outputs(*encoder_outputs*, *num_pos*, *batch_size*, *num_size*, *hidden_size*)

model_test(*batch_data: dict*) → Tuple[str, list, list]

Model test.

Parameters **batch_data** – one batch data.

Returns result_type, predicted equation, target equation.

batch_data should include keywords 'input1', 'input2', 'output1', 'output2', 'input1 len', 'parse graph', 'num stack', 'num pos', 'num order', 'num list'

predict(*batch_data*, *output_all_layers=False*)

training: bool

tree_decoder_forward(*encoder_outputs*, *problem_output*, *all_nums_encoder_outputs*, *nums_stack*, *seq_mask*, *num_mask*, *target=None*, *output_all_layers=False*)

5.4 mwptoolkit.model.PreTrain

5.4.1 mwptoolkit.model.PreTrain.bertgen

class mwptoolkit.model.PreTrain.bertgen.**BERTGen**(*config*, *dataset*)

Bases: torch.nn.modules.module.Module

Reference: Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding".

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

calculate_loss(*batch_data: dict*) → float

Finish forward-propagating, calculating loss and back-propagation.

Parameters **batch_data** (*dict*) – one batch data.

Returns loss value.

Return type float

convert_idx2symbol(*outputs*, *num_lists*)

decode(*output*)

decode_(*outputs*)

decoder_forward(*encoder_outputs*, *source_padding_mask*, *target=None*, *output_all_layers=None*)

encoder_forward(*seq*, *output_all_layers=False*)

forward(*seq*, *target=None*, *output_all_layers=False*) → Tuple[torch.Tensor, torch.Tensor, Dict[str, Any]]

Parameters

- **seq** (*torch.Tensor*) – input sequence, shape: [batch_size, seq_length].
- **target** (*torch.Tensor | None*) – target, shape: [batch_size,target_length].
- **output_all_layers** (*bool*) – return output of all layers if output_all_layers is True, default False.

Returns token_logits: [batch_size, output_length, output_size], symbol_outputs: [batch_size,output_length], model_all_outputs.

Return type tuple(torch.Tensor, torch.Tensor, dict)

model_test(*batch_data: dict*) → tuple

Model test.

Parameters **batch_data** (*dict*) – one batch data.

Returns predicted equation, target equation.

Return type tuple(list,list)

predict(*batch_data: dict*, *output_all_layers=False*)

predict samples without target.

Parameters

- **batch_data** (*dict*) – one batch data.
- **output_all_layers** (*bool*) – return all layer outputs of model.

Returns token_logits, symbol_outputs, all_layer_outputs

training: bool

5.4.2 mwptoolkit.model.PreTrain.gpt2

class mwptoolkit.model.PreTrain.gpt2.**GPT2**(*config*, *dataset*)

Bases: torch.nn.modules.module.Module

Reference: Radford et al. "Language Models are Unsupervised Multitask Learners".

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

calculate_loss(*batch_data: dict*) → float

Finish forward-propagating, calculating loss and back-propagation.

Parameters **batch_data** (*dict*) – one batch data.

Returns loss value.

Return type float

convert_idx2symbol(*outputs*, *num_lists*)

decode_(*outputs*)

decoder_forward(*seq*, *target=None*, *output_all_layers=False*)

encode_(*inputs*)

forward(*seq*, *target=None*, *output_all_layers=False*) → Tuple[torch.Tensor, torch.Tensor, Dict[str, Any]]

Parameters

- **seq** (*torch.Tensor*) – input sequence, shape: [batch_size, seq_length].
- **target** (*torch.Tensor | None*) – target, shape: [batch_size,target_length].
- **output_all_layers** (*bool*) – return output of all layers if output_all_layers is True, default False.

Returns token_logits: [batch_size, output_length, output_size], symbol_outputs: [batch_size,output_length], model_all_outputs.

Return type tuple(torch.Tensor, torch.Tensor, dict)

list2str(*x*)

model_test(*batch_data: dict*) → tuple

Model test.

Parameters **batch_data** (*dict*) – one batch data.

Returns predicted equation, target equation.

Return type tuple(list,list)

predict(*batch_data: dict*, *output_all_layers=False*)

predict samples without target.

Parameters

- **batch_data** (*dict*) – one batch data.
- **output_all_layers** (*bool*) – return all layer outputs of model.

Returns token_logits, symbol_outputs, all_layer_outputs

training: bool

5.4.3 mwptoolkit.model.PreTrain.robertagen

class mwptoolkit.model.PreTrain.robertagen.**RobertaGen**(*config*, *dataset*)

Bases: torch.nn.modules.module.Module

Reference: Liu et al. "RoBERTa: A Robustly Optimized BERT Pretraining Approach".

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

calculate_loss(*batch_data: dict*) → float

Finish forward-propagating, calculating loss and back-propagation.

Parameters **batch_data** (*dict*) – one batch data.

Returns loss value.

Return type float

convert_idx2symbol(*outputs*, *num_lists*)

decode(*output*)

decode_(*outputs*)

decoder_forward(*encoder_outputs*, *source_padding_mask*, *target=None*, *output_all_layers=None*)

encoder_forward(*seq*, *output_all_layers=False*)

forward(*seq*, *target=None*, *output_all_layers=False*) → Tuple[torch.Tensor, torch.Tensor, Dict[str, Any]]

Parameters

- **seq** (*torch.Tensor*) – input sequence, shape: [batch_size, seq_length].
- **target** (*torch.Tensor | None*) – target, shape: [batch_size,target_length].
- **output_all_layers** (*bool*) – return output of all layers if output_all_layers is True, default False.

Returns token_logits: [batch_size, output_length, output_size], symbol_outputs: [batch_size,output_length], model_all_outputs.

Return type tuple(torch.Tensor, torch.Tensor, dict)

model_test(*batch_data: dict*) → tuple

Model test.

Parameters **batch_data** (*dict*) – one batch data.

Returns predicted equation, target equation.

Return type tuple(list,list)

predict(*batch_data: dict*, *output_all_layers=False*)

predict samples without target.

Parameters

- **batch_data** (*dict*) – one batch data.
- **output_all_layers** (*bool*) – return all layer outputs of model.

Returns token_logits, symbol_outputs, all_layer_outputs

training: bool

CHAPTER

SIX

MWPTOOLKIT.MODULE

6.1 mwptoolkit.module.Attention

6.1.1 mwptoolkit.module.Attention.group_attention

class mwptoolkit.module.Attention.group_attention.**GroupAttention**(*h*, *d_model*, *dropout=0.1*)

Bases: torch.nn.modules.module.Module

Take in model size and number of heads.

forward(*query*, *key*, *value*, *mask=None*)

Parameters

- **query** (*torch.Tensor*) – shape [batch_size, head_nums, sequence_length, dim_k].
- **key** (*torch.Tensor*) – shape [batch_size, head_nums, sequence_length, dim_k].
- **value** (*torch.Tensor*) – shape [batch_size, head_nums, sequence_length, dim_k].
- **mask** (*torch.Tensor*) – group attention mask, shape [batch_size, head_nums, sequence_length, sequence_length].

Returns shape [batch_size, sequence_length, hidden_size].

Return type torch.Tensor

get_mask(*src*, *split_list*, *pad=0*)

Parameters

- **src** (*torch.Tensor*) – source sequence, shape [batch_size, sequence_length].
- **split_list** (*list*) – group split index.
- **pad** (*int*) – pad token index.

Returns group attention mask, shape [batch_size, 4, sequence_length, sequence_length].

Return type torch.Tensor

src_to_mask(*src*, *split_list*)

training: bool

mwptoolkit.module.Attention.group_attention.**attention**(*query*, *key*, *value*, *mask=None*, *dropout=None*)

Compute Scaled Dot Product Attention

Parameters

- **query** (*torch.Tensor*) – shape [batch_size, sequence_length, hidden_size].
- **key** (*torch.Tensor*) – shape [batch_size, sequence_length, hidden_size].
- **value** (*torch.Tensor*) – shape [batch_size, sequence_length, hidden_size].
- **mask** (*torch.Tensor*) – group attention mask, shape [batch_size, 4, sequence_length, sequence_length].

Return type tuple(torch.Tensor, torch.Tensor)

mwptoolkit.module.Attention.group_attention.**group_mask**(*batch*, *type='self'*, *pad=0*)

mwptoolkit.module.Attention.group_attention.**src_to_mask**(*src*, *vocab_dict*)

6.1.2 mwptoolkit.module.Attention.multi_head_attention

class mwptoolkit.module.Attention.multi_head_attention.**EPTMultiHeadAttention**(***config*)

Bases: torch.nn.modules.module.Module

Class for computing multi-head attention (follows the paper, 'Attention is all you need')

This class computes attention over K-V pairs with query Q, i.e.

Initialize MultiHeadAttention class

Keyword Arguments

- **hidden_dim** (*int*) – Vector dimension of hidden states (H). 768 by default
- **num_heads** (*int*) – Number of attention heads (N). 12 by default
- **dropout_p** (*float*) – Probability of dropout. 0 by default

forward(*query: torch.Tensor*, *key_value: Optional[torch.Tensor] = None*, *key_ignorance_mask: Optional[torch.Tensor] = None*, *attention_mask: Optional[torch.Tensor] = None*, *return_weights: bool = False*, ***kwargs*)

Compute multi-head attention

Parameters

- **query** (*torch.Tensor*) – FloatTensor representing the query matrix with shape [batch_size, query_sequence_length, hidden_size].
- **key_value** (*torch.Tensor*) – FloatTensor representing the key matrix or value matrix with shape [batch_size, key_sequence_length, hidden_size] or [1, key_sequence_length, hidden_size]. By default, this is *None* (Use query matrix as a key matrix).
- **key_ignorance_mask** (*torch.Tensor*) – BoolTensor representing the mask for ignoring column vector in key matrix, with shape [batch_size, key_sequence_length]. If an element at (b, t) is *True,* then all return elements at batch_size=b, key_sequence_length=t will set to be -Infinity. By default, this is *None* (There's no mask to apply).
- **attention_mask** (*torch.Tensor*) – BoolTensor representing Attention mask for ignoring a key for each query item, with shape [query_sequence_length, key_sequence_length]. If an element at (s, t) is *True,* then all return elements at query_sequence_length=s, key_sequence_length=t will set to be -Infinity. By default, this is *None* (There's no mask to apply).
- **return_weights** (*bool*) – Use *True* to return attention weights. By default, this is *True.*

Returns If head_at_last is True, return (Attention Output, Attention Weights). Otherwise, return only the Attention Output. Attention Output: Shape [batch_size, query_sequence_length, hidden_size]. Attention Weights: Shape [batch_size, query_sequence_length, key_sequence_length, head_nums].

Return type Union[torch.FloatTensor, Tuple[torch.FloatTensor, torch.FloatTensor]]

training: bool

class mwptoolkit.module.Attention.multi_head_attention.**EPTMultiHeadAttentionWeights**(***config*)

Bases: torch.nn.modules.module.Module

Class for computing multi-head attention weights (follows the paper, 'Attention is all you need')

This class computes dot-product between query Q and key K, i.e.

Initialize MultiHeadAttentionWeights class

Keyword Arguments

- **hidden_dim** (*int*) – Vector dimension of hidden states (H). 768 by default.
- **num_heads** (*int*) – Number of attention heads (N). 12 by default.

forward(*query: torch.Tensor*, *key: Optional[torch.Tensor] = None*, *key_ignorance_mask: Optional[torch.Tensor] = None*, *attention_mask: Optional[torch.Tensor] = None*, *head_at_last: bool = True*) → torch.Tensor

Compute multi-head attention weights

Parameters

- **query** (*torch.Tensor*) – FloatTensor representing the query matrix with shape [batch_size, query_sequence_length, hidden_size].
- **key** (*torch.Tensor*) – FloatTensor representing the key matrix with shape [batch_size, key_sequence_length, hidden_size] or [1, key_sequence_length, hidden_size]. By default, this is *None* (Use query matrix as a key matrix)
- **key_ignorance_mask** (*torch.Tensor*) – BoolTensor representing the mask for ignoring column vector in key matrix, with shape [batch_size, key_sequence_length]. If an element at (b, t) is *True,* then all return elements at batch_size=b, key_sequence_length=t will set to be -Infinity. By default, this is *None* (There's no mask to apply).
- **attention_mask** (*torch.Tensor*) – BoolTensor representing Attention mask for ignoring a key for each query item, with shape [query_sequence_length, key_sequence_length]. If an element at (s, t) is *True,* then all return elements at sequence_length=s, T=t will set to be -Infinity. By default, this is *None* (There's no mask to apply).
- **head_at_last** (*bool*) – Use *True* to make shape of return value be [batch_size, query_sequence_length, key_sequence_length, head_nums]. If *False,* this method will return [batch_size, head_nums, sequence_length, key_sequence_length]. By default, this is *True*

Returns FloatTensor of Multi-head Attention weights.

Return type torch.FloatTensor

property hidden_dim: int

int :return: Vector dimension of hidden states (H)

Type rtype

property num_heads: int

int :return: Number of attention heads (N)

Type rtype

training: bool

class mwptoolkit.module.Attention.multi_head_attention.**MultiHeadAttention**(*embedding_size*, *num_heads*, *dropout_ratio=0.0*)

Bases: torch.nn.modules.module.Module

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

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

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

Multi-head attention

Parameters

- **query** (*torch.Tensor*) – shape [batch_size, tgt_len, embedding_size].
- **key** (*torch.Tensor*) – shape [batch_size, src_len, embedding_size].
- **value** (*torch.Tensor*) – shape [batch_size, src_len, embedding_size].
- **key_padding_mask** (*torch.Tensor*) – shape [batch_size, src_len].
- **attn_mask** (*torch.BoolTensor*) – 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(torch.Tensor, torch.Tensor)

training: bool

6.1.3 mwptoolkit.module.Attention.self_attention

class mwptoolkit.module.Attention.self_attention.**SelfAttention**(*hidden_size*)

Bases: torch.nn.modules.module.Module

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

forward(*inputs*)

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 mwptoolkit.module.Attention.self_attention.**SelfAttentionMask**(*init_size=100*)

Bases: torch.nn.modules.module.Module

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

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

6.1.4 mwptoolkit.module.Attention.seq_attention

class mwptoolkit.module.Attention.seq_attention.**Attention**(*dim_value*, *dim_query*, *dim_hidden=256*, *dropout_rate=0.5*)

Bases: torch.nn.modules.module.Module

Calculate attention

Parameters

- **dim_value** (*int*) – Dimension of value.
- **dim_query** (*int*) – Dimension of query.
- **dim_hidden** (*int*) – Dimension of hidden layer in attention calculation.

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

forward(*value*, *query*, *lens*)

Generate variable embedding with attention.

Parameters

- **query** (*FloatTensor*) – Current hidden state, with size [batch_size, dim_query].
- **value** (*FloatTensor*) – Sequence to be attented, with size [batch_size, seq_len, dim_value].
- **lens** (*list of int*) – Lengths of values in a batch.

Returns Calculated attention, with size [batch_size, dim_value].

Return type FloatTensor

training: bool

class mwptoolkit.module.Attention.seq_attention.**MaskedRelevantScore**(*dim_value*, *dim_query*, *dim_hidden=256*, *dropout_rate=0.0*)

Bases: torch.nn.modules.module.Module

Relevant score masked by sequence lengths.

Parameters

- **dim_value** (*int*) – Dimension of value.
- **dim_query** (*int*) – Dimension of query.
- **dim_hidden** (*int*) – Dimension of hidden layer in attention calculation.

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

forward(*value*, *query*, *lens*)

Choose candidate from candidates.

Parameters

- **query** (*torch.FloatTensor*) – Current hidden state, with size [batch_size, dim_query].
- **value** (*torch.FloatTensor*) – Sequence to be attented, with size [batch_size, seq_len, dim_value].
- **lens** (*list of int*) – Lengths of values in a batch.

Returns Activation for each operand, with size [batch, max([len(os) for os in operands])].

Return type torch.Tensor

training: **bool**

class mwptoolkit.module.Attention.seq_attention.**RelevantScore**(*dim_value*, *dim_query*, *hidden1*, *dropout_rate=0*)

Bases: torch.nn.modules.module.Module

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

forward(*value*, *query*)

Parameters

- **value** (*torch.FloatTensor*) – shape [batch, seq_len, dim_value].
- **query** (*torch.FloatTensor*) – shape [batch, dim_query].

training: **bool**

class mwptoolkit.module.Attention.seq_attention.**SeqAttention**(*hidden_size*, *context_size*)

Bases: torch.nn.modules.module.Module

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

forward(*inputs*, *encoder_outputs*, *mask*)

Parameters

- **inputs** (*torch.Tensor*) – shape [batch_size, 1, hidden_size].
- **encoder_outputs** (*torch.Tensor*) – shape [batch_size, sequence_length, hidden_size].

Returns output, shape [batch_size, 1, context_size]. attention, shape [batch_size, 1, sequence_length].

Return type tuple(torch.Tensor, torch.Tensor)

training: **bool**

6.1.5 mwptoolkit.module.Attention.tree_attentio

class mwptoolkit.module.Attention.tree_attention.**TreeAttention**(*input_size*, *hidden_size*)

Bases: torch.nn.modules.module.Module

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

forward(*hidden*, *encoder_outputs*, *seq_mask=None*)

Parameters

- **hidden** (*torch.Tensor*) – hidden representation, shape [1, batch_size, hidden_size]
- **encoder_outputs** (*torch.Tensor*) – output from encoder, shape [sequence_length, batch_size, hidden_size].
- **seq_mask** (*torch.Tensor*) – sequence mask, shape [batch_size, sequence_length].

Returns attention energies, shape [batch_size, 1, sequence_length].

Return type attn_energies (torch.Tensor)

training: bool

6.2 mwptoolkit.module.Decoder

6.2.1 mwptoolkit.module.Decoder.ept_decoder

class mwptoolkit.module.Decoder.ept_decoder.**AveragePooling**(*dim: int = - 1*, *keepdim: bool = False*)

Bases: torch.nn.modules.module.Module

Layer class for computing mean of a sequence

Layer class for computing mean of a sequence

Parameters

- **dim** (*int*) – Dimension to be averaged. -1 by default.
- **keepdim** (*bool*) – True if you want to keep averaged dimensions. False by default.

extra_repr()

Set the extra representation of the module

To print customized extra information, you should re-implement this method in your own modules. Both single-line and multi-line strings are acceptable.

forward(*tensor: torch.Tensor*)

Do average pooling over a sequence

Parameters **tensor** (*torch.Tensor*) – FloatTensor to be averaged.

Returns Averaged result.

Return type torch.FloatTensor

training: bool

class mwptoolkit.module.Decoder.ept_decoder.**DecoderModel**(*config*)

Bases: torch.nn.modules.module.Module

Base model for equation generation/classification (Abstract class)

Initiate Equation Builder instance

Parameters **config** (*ModelConfig*) – Configuration of this model

_build_target_dict(***kwargs*) → Dict[str, torch.Tensor]

Build dictionary of target matrices.

Return type Dict[str, torch.Tensor]

Returns Dictionary of target values

_forward_single(***kwargs*) → Dict[str, torch.Tensor]

Forward computation of a single beam

Return type Dict[str, torch.Tensor]

Returns Dictionary of computed values

_init_weights(*module: torch.nn.modules.module.Module*)

Initialize weights

Parameters **module** (*nn.Module*) – Module to be initialized.

forward(*text: Optional[torch.Tensor] = None*, *text_pad: Optional[torch.Tensor] = None*, *text_num: Optional[torch.Tensor] = None*, *text_numpad: Optional[torch.Tensor] = None*, *equation: Optional[torch.Tensor] = None*, *beam: int = 1*, *max_len: int = 128*, *function_arities: Optional[Dict[int, int]] = None*)

Forward computation of decoder model

Returns

Dictionary of tensors. If this model is currently on training phase, values will be accuracy or loss tensors Otherwise, values will be tensors representing predicted distribution of output

Return type Dict[str, torch.Tensor]

init_factor()

Returns Standard deviation of normal distribution that will be used for initializing weights.

Return type float

property is_expression_type: bool

bool :return: True if this model requires Expression type sequence

Type rtype

property required_field: str

str :return: Name of required field type to process

Type rtype

training: bool

class mwptoolkit.module.Decoder.ept_decoder.**ExpressionDecoderModel**(*config*, *out_opsym2idx*, *out_idx2opsym*, *out_consym2idx*, *out_idx2consym*)

Bases: *mwptoolkit.module.Decoder.ept_decoder.DecoderModel*

Decoding model that generates expression sequences (Abstract class)

Initiate Equation Builder instance

Parameters **config** (*ModelConfig*) – Configuration of this model

_build_decoder_context(*embedding: torch.Tensor*, *embedding_pad: Optional[torch.Tensor] = None*, *text: Optional[torch.Tensor] = None*, *text_pad: Optional[torch.Tensor] = None*)

Compute decoder's hidden state vectors

Parameters

- **embedding** (*torch.Tensor*) – FloatTensor containing input vectors. Shape [batch_size, equation_length, hidden_size],
- **embedding_pad** (*torch.Tensor*) – BoolTensor, whose values are True if corresponding position is PAD in the decoding sequence, Shape [batch_size, equation_length]
- **text** (*torch.Tensor*) – FloatTensor containing encoder's hidden states. Shape [batch_size, input_sequence_length, hidden_size].
- **text_pad** (*torch.Tensor*) – BoolTensor, whose values are True if corresponding position is PAD in the input sequence. Shape [batch_size, input_sequence_length]

Returns A FloatTensor of shape [batch_size, equation_length, hidden_size], which contains decoder's hidden states.

Return type torch.Tensor

_build_decoder_input(*ids: torch.Tensor*, *nums: torch.Tensor*)

Compute input of the decoder

Parameters

- **ids** (*torch.Tensor*) – LongTensor containing index-type information of an operator and its operands. Shape: [batch_size, equation_length, 1+2*arity_size]
- **nums** (*torch.Tensor*) – FloatTensor containing encoder's hidden states corresponding to numbers in the text. Shape: [batch_size, num_size, hidden_size].

Returns A FloatTensor representing input vector. Shape [batch_size, equation_length, hidden_size].

Return type torch.Tensor

_build_operand_embed(*ids: torch.Tensor*, *mem_pos: torch.Tensor*, *nums: torch.Tensor*) → torch.Tensor

Build operand embedding a_ij in the paper.

Parameters

- **ids** (*torch.Tensor*) – LongTensor containing index-type information of operands. (This corresponds to a_ij in the paper)
- **mem_pos** (*torch.Tensor*) – FloatTensor containing positional encoding used so far. (i.e. PE(.) in the paper)
- **nums** (*torch.Tensor*) – FloatTensor containing encoder's hidden states corresponding to numbers in the text. (i.e. e_{a_ij} in the paper)

Return type torch.Tensor

Returns A FloatTensor representing operand embedding vector a_ij in Equation 3, 4, 5

_forward_single(*text: Optional[torch.Tensor] = None*, *text_pad: Optional[torch.Tensor] = None*, *text_num: Optional[torch.Tensor] = None*, *text_numpad: Optional[torch.Tensor] = None*, *equation: Optional[torch.Tensor] = None*)

Forward computation of a single beam

Parameters

- **text** (*torch.Tensor*) – FloatTensor containing encoder's hidden states. Shape [batch_size, input_sequence_length, hidden_size].
- **text_pad** (*torch.Tensor*) – BoolTensor, whose values are True if corresponding position is PAD in the input sequence. Shape [batch_size, input_sequence_length]
- **text_num** (*torch.Tensor*) – FloatTensor containing encoder's hidden states corresponding to numbers in the text. Shape: [batch_size, num_size, hidden_size].
- **equation** (*torch.Tensor*) – LongTensor containing index-type information of an operator and its operands. Shape: [batch_size, equation_length, 1+2*arity_size].

Returns

Dictionary of followings 'operator': Log probability of next operators. FloatTensor with shape [batch_size, equation_length, operator_size]. '_out': Decoder's hidden states. FloatTensor with shape [batch_size, equation_length, hidden_size]. '_not_usable': Indicating positions that corresponding output values are not usable in the operands. BoolTensor with Shape [batch_size, equation_length].

Return type Dict[str, torch.Tensor]

embed_to_hidden

Transformer layer

function_arities

Embedding layers

operand_norm

Linear Transformation

operand_source_embedding

Scalar parameters

operand_source_factor

Layer Normalizations

shared_decoder_layer

Output layer

training: bool

class mwptoolkit.module.Decoder.ept_decoder.**ExpressionPointerTransformer**(*config*, *out_opsym2idx*, *out_idx2opsym*, *out_consym2idx*, *out_idx2consym*)

Bases: *mwptoolkit.module.Decoder.ept_decoder.ExpressionDecoderModel*

The EPT model

Initiate Equation Builder instance

Parameters **config** (*ModelConfig*) – Configuration of this model

_build_attention_keys(*num: torch.Tensor*, *mem: torch.Tensor*, *num_pad: Optional[torch.Tensor] = None*, *mem_pad: Optional[torch.Tensor] = None*)

Generate Attention Keys by concatenating all items.

Parameters

- **num** (*torch.Tensor*) – FloatTensor containing encoder's hidden states corresponding to numbers in the text. Shape [batch_size, num_size, hidden_size].
- **mem** (*torch.Tensor*) – FloatTensor containing decoder's hidden states corresponding to prior expression outputs. Shape [batch_size, equation_length, hidden_size].
- **num_pad** (*torch.Tensor*) – BoolTensor, whose values are True if corresponding position is PAD in the number sequence. Shape [batch_size, num_size]
- **mem_pad** (*torch.Tensor*) – BoolTensor, whose values are True if corresponding position is PAD in the target expression sequence. Shape [batch_size, equation_length]

Returns

Triple of Tensors

- [0] Keys (A_ij in the paper). Shape [batch_size, constant_size+num_size+equation_length, hidden_size], where C = size of constant vocabulary.
- [1] Mask for positions that should be ignored in keys. Shape [batch_size, C+num_size+equation_length]
- [2] Forward Attention Mask to ignore future tokens in the expression sequence. Shape [equation_length, C+num_size+equation_length]

Return type Tuple[torch.Tensor, torch.Tensor, torch.Tensor]

_build_operand_embed(*ids: torch.Tensor*, *mem_pos: torch.Tensor*, *nums: torch.Tensor*)

Build operand embedding.

Parameters

- **ids** (*torch.Tensor*) – LongTensor containing source-content information of operands. Shape [batch_size, equation_length, 1+2*arity_size].
- **mem_pos** (*torch.Tensor*) – FloatTensor containing positional encoding used so far. Shape [batch_size, equation_length, hidden_size].
- **nums** (*torch.Tensor*) – FloatTensor containing encoder's hidden states corresponding to numbers in the text. Shape [batch_size, num_size, hidden_size].

Returns A FloatTensor representing operand embedding vector. Shape [batch_size, equation_length, arity_size, hidden_size]

Return type torch.Tensor

_build_target_dict(*equation*, *num_pad=None*)

Build dictionary of target matrices.

Returns

Dictionary of target values 'operator': Index of next operators. LongTensor with shape [batch_size, equation_length]. 'operand_J': Index of next J-th operands. LongTensor with shape [batch_size, equation_length].

Return type Dict[str, torch.Tensor]

_forward_single(*text: Optional[torch.Tensor] = None*, *text_pad: Optional[torch.Tensor] = None*, *text_num: Optional[torch.Tensor] = None*, *text_numpad: Optional[torch.Tensor] = None*, *equation: Optional[torch.Tensor] = None*)

Forward computation of a single beam

Parameters

- **text** (*torch.Tensor*) – FloatTensor containing encoder's hidden states. Shape [batch_size, input_sequence_length, hidden_size].
- **text_pad** (*torch.Tensor*) – BoolTensor, whose values are True if corresponding position is PAD in the input sequence. Shape [batch_size, input_sequence_length]
- **text_num** (*torch.Tensor*) – FloatTensor containing encoder's hidden states corresponding to numbers in the text. Shape: [batch_size, num_size, hidden_size].
- **text_numpad** (*torch.Tensor*) – BoolTensor, whose values are True if corresponding position is PAD in the number sequence. Shape [batch_size, num_size]
- **equation** (*torch.Tensor*) – LongTensor containing index-type information of an operator and its operands. Shape: [batch_size, equation_length, 1+2*arity_size].

Returns

Dictionary of followings 'operator': Log probability of next operators.FloatTensor with shape [batch_size, equation_length, operator_size]. 'operand_J': Log probability of next J-th operands. FloatTensor with shape [batch_size, equation_length, operand_size].

Return type Dict[str, torch.Tensor]

constant_word_embedding

Output layer

operand_out

Initialize weights

property required_field: str

str :return: Name of required field type to process

Type rtype

training: bool

class mwptoolkit.module.Decoder.ept_decoder.**ExpressionTransformer**(*config*, *out_opsym2idx*, *out_idx2opsym*, *out_consym2idx*, *out_idx2consym*)

Bases: *mwptoolkit.module.Decoder.ept_decoder.ExpressionDecoderModel*

Vanilla Transformer + Expression (The second ablated model)

Initiate Equation Builder instance

Parameters **config** (*ModelConfig*) – Configuration of this model

_build_operand_embed(*ids: torch.Tensor*, *mem_pos: torch.Tensor*, *nums: torch.Tensor*)

Build operand embedding.

Parameters

- **ids** (*torch.Tensor*) – LongTensor containing source-content information of operands. Shape [batch_size, equation_length, 1+2*arity_size].

- **mem_pos** (*torch.Tensor*) – FloatTensor containing positional encoding used so far. Shape [batch_size, equation_length, hidden_size], where hidden_size = dimension of hidden state
- **nums** (*torch.Tensor*) – FloatTensor containing encoder's hidden states corresponding to numbers in the text. Shape [batch_size, num_size, hidden_size].

Returns A FloatTensor representing operand embedding vector. Shape [batch_size, equation_length, arity_size, hidden_size]

Return type torch.Tensor

_build_target_dict(*equation*, *num_pad=None*)

Build dictionary of target matrices.

Returns

Dictionary of target values 'operator': Index of next operators. LongTensor with shape [batch_size, equation_length]. 'operand_J': Index of next J-th operands. LongTensor with shape [batch_size, equation_length].

Return type Dict[str, torch.Tensor]

_forward_single(*text: Optional[torch.Tensor] = None*, *text_pad: Optional[torch.Tensor] = None*, *text_num: Optional[torch.Tensor] = None*, *text_numpad: Optional[torch.Tensor] = None*, *equation: Optional[torch.Tensor] = None*)

Forward computation of a single beam

Parameters

- **text** (*torch.Tensor*) – FloatTensor containing encoder's hidden states. Shape [batch_size, input_sequence_length, hidden_size].
- **text_pad** (*torch.Tensor*) – BoolTensor, whose values are True if corresponding position is PAD in the input sequence. Shape [batch_size, input_sequence_length]
- **text_num** (*torch.Tensor*) – FloatTensor containing encoder's hidden states corresponding to numbers in the text.Shape: [batch_size, num_size, hidden_size].
- **text_numpad** (*torch.Tensor*) – BoolTensor, whose values are True if corresponding position is PAD in the number sequence. Shape [batch_size, num_size]
- **equation** (*torch.Tensor*) – LongTensor containing index-type information of an operator and its operands. Shape: [batch_size, equation_length, 1+2*arity_size].

Returns

Dictionary of followings 'operator': Log probability of next operators. FloatTensor with shape [batch_size, equation_length, operator_size], where operator_size = size of operator vocabulary. 'operand_J': Log probability of next J-th operands.FloatTensor with shape [batch_size, equation_length, operand_size].

Return type Dict[str, torch.Tensor]

operand_out

Initialize weights

operand_word_embedding

Output layer

property required_field: str

str :return: Name of required field type to process

Type rtype

training: bool

class mwptoolkit.module.Decoder.ept_decoder.**LogSoftmax**(*dim: Optional[int] = None*)

Bases: torch.nn.modules.activation.LogSoftmax

LogSoftmax layer that can handle infinity values.

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

dim: Optional[int]

forward(*tensor: torch.Tensor*)

Compute log(softmax(tensor))

Parameters **torch.Tensor** (*tensor*) – FloatTensor whose log-softmax value will be computed

Returns LogSoftmax result.

Return type torch.FloatTensor

class mwptoolkit.module.Decoder.ept_decoder.**OpDecoderModel**(*config*)

Bases: *mwptoolkit.module.Decoder.ept_decoder.DecoderModel*

Decoding model that generates Op(Operator/Operand) sequences (Abstract class)

Initiate Equation Builder instance

Parameters **config** (*ModelConfig*) – Configuration of this model

_build_decoder_context(*embedding: torch.Tensor*, *embedding_pad: Optional[torch.Tensor] = None*, *text: Optional[torch.Tensor] = None*, *text_pad: Optional[torch.Tensor] = None*)

Compute decoder's hidden state vectors.

Parameters

- **embedding** (*torch.Tensor*) – FloatTensor containing input vectors. Shape [batch_size, decoding_sequence, input_embedding_size].
- **embedding_pad** (*torch.Tensor*) – BoolTensor, whose values are True if corresponding position is PAD in the decoding sequence. Shape [batch_size, decoding_sequence]
- **text** (*torch.Tensor*) – FloatTensor containing encoder's hidden states. Shape [batch_size, input_sequence_length, input_embedding_size].
- **text_pad** (*torch.Tensor*) – BoolTensor, whose values are True if corresponding position is PAD in the input sequence. Shape [batch_size, input_sequence_length]

Returns: torch.Tensor: A FloatTensor of shape [batch_size, decoding_sequence, hidden_size], which contains decoder's hidden states.

_build_decoder_input(*ids: torch.Tensor*, *nums: torch.Tensor*)

Compute input of the decoder.

Parameters

- **ids** (*torch.Tensor*) – LongTensor containing op tokens. Shape: [batch_size, equation_length]
- **nums** (*torch.Tensor*) – FloatTensor containing encoder's hidden states corresponding to numbers in the text. Shape: [batch_size, num_size, hidden_size],

Returns A FloatTensor representing input vector. Shape [batch_size, equation_length, hidden_size].

Return type torch.Tensor

_build_word_embed(*ids: torch.Tensor*, *nums: torch.Tensor*)

Build Op embedding

Parameters

- **ids** (*torch.Tensor*) – LongTensor containing source-content information of operands. Shape [batch_size, equation_length].
- **nums** (*torch.Tensor*) – FloatTensor containing encoder's hidden states corresponding to numbers in the text. Shape [batch_size, num_size, hidden_size].

Returns A FloatTensor representing op embedding vector. Shape [batch_size, equation_length, hidden_size]

Return type torch.Tensor

_forward_single(*text: Optional[torch.Tensor] = None*, *text_pad: Optional[torch.Tensor] = None*, *text_num: Optional[torch.Tensor] = None*, *text_numpad: Optional[torch.Tensor] = None*, *equation: Optional[torch.Tensor] = None*)

Forward computation of a single beam

Parameters

- **text** (*torch.Tensor*) – FloatTensor containing encoder's hidden states e_i. Shape [batch_size, input_sequence_length, input_embedding_size].
- **text_pad** (*torch.Tensor*) – BoolTensor, whose values are True if corresponding position is PAD in the input sequence. Shape [batch_size, input_sequence_length]
- **text_num** (*torch.Tensor*) – FloatTensor containing encoder's hidden states corresponding to numbers in the text. Shape: [batch_size, num_size, input_embedding_size].
- **equation** (*torch.Tensor*) – LongTensor containing index-type information of an operator and its operands. Shape: [batch_size, equation_length, 1+2*arity_size].

Returns

Dictionary of followings '_out': Decoder's hidden states. FloatTensor with shape [batch_size, equation_length, hidden_size].

Return type Dict[str, torch.Tensor]

pos_factor

Decoding layer

training: bool

class mwptoolkit.module.Decoder.ept_decoder.**Squeeze**(*dim: int = - 1*)

Bases: torch.nn.modules.module.Module

Layer class for squeezing a dimension

Layer class for squeezing a dimension

Parameters **dim** (*int*) – Dimension to be squeezed, -1 by default.

extra_repr()

Set the extra representation of the module

To print customized extra information, you should re-implement this method in your own modules. Both single-line and multi-line strings are acceptable.

forward(*tensor: torch.Tensor*)

Do squeezing

Parameters **tensor** (*torch.Tensor*) – FloatTensor to be squeezed.

Returns Squeezed result.

Return type torch.FloatTensor

training: bool

class mwptoolkit.module.Decoder.ept_decoder.**VanillaOpTransformer**(*config*)

Bases: *mwptoolkit.module.Decoder.ept_decoder.OpDecoderModel*

The vanilla Transformer model

Initiate Equation Builder instance

Parameters **config** (*ModelConfig*) – Configuration of this model

_build_target_dict(*equation*, *num_pad=None*)

Build dictionary of target matrices.

Returns

Dictionary of target values 'op': Index of next op tokens. LongTensor with shape [batch_size, equation_length].

Return type Dict[str, torch.Tensor]

_build_word_embed(*ids: torch.Tensor*, *nums: torch.Tensor*)

Build Op embedding

Parameters

- **ids** (*torch.Tensor*) – LongTensor containing source-content information of operands. Shape [batch_size, equation_length].
- **nums** (*torch.Tensor*) – FloatTensor containing encoder's hidden states corresponding to numbers in the text. Shape [batch_size, num_size, hidden_size].

Returns A FloatTensor representing op embedding vector. Shape [batch_size, equation_length, hidden_size].

Return type torch.Tensor

_forward_single(*text: Optional[torch.Tensor] = None*, *text_pad: Optional[torch.Tensor] = None*, *text_num: Optional[torch.Tensor] = None*, *text_numpad: Optional[torch.Tensor] = None*, *equation: Optional[torch.Tensor] = None*)

Forward computation of a single beam

Parameters

- **text** (*torch.Tensor*) – FloatTensor containing encoder's hidden states. Shape [batch_size, input_sequence_length, input_embedding_size].
- **text_pad** (*torch.Tensor*) – BoolTensor, whose values are True if corresponding position is PAD in the input sequence. Shape [batch_size, input_sequence_length]
- **text_num** (*torch.Tensor*) – FloatTensor containing encoder's hidden states corresponding to numbers in the text. Shape: [batch_size, num_size, input_embedding_size].
- **equation** (*torch.Tensor*) – LongTensor containing index-type information of an operator and its operands. Shape: [batch_size, equation_length].

Returns

Dictionary of followings 'op': Log probability of next op tokens. FloatTensor with shape [batch_size, equation_length, operator_size].

Return type Dict[str, torch.Tensor]

property required_field: str

str :return: Name of required field type to process

Type rtype

softmax

Initialize weights

training: bool

mwptoolkit.module.Decoder.ept_decoder.**apply_across_dim**(*function*, *dim=1*, *shared_keys=None*, ***tensors*)

Apply a function repeatedly for each tensor slice through the given dimension. For example, we have tensor [batch_size, X, input_sequence_length] and dim = 1, then we will concatenate the following matrices on dim=1. - function([:, 0, :]) - function([:, 1, :]) - ... - function([:, X-1, :]).

Parameters

- **function** (*function*) – Function to apply.
- **dim** (*int*) – Dimension through which we'll apply function. (1 by default)
- **shared_keys** (*set*) – Set of keys representing tensors to be shared. (None by default)
- **tensors** (*torch.Tensor*) – Keyword arguments of tensors to compute. Dimension should >= *dim*.

Returns Dictionary of tensors, whose keys are corresponding to the output of the function.

Return type Dict[str, torch.Tensor]

mwptoolkit.module.Decoder.ept_decoder.**apply_module_dict**(*modules: torch.nn.modules.container.ModuleDict*, *encoded: torch.Tensor*, ***kwargs*)

Predict next entry using given module and equation.

Parameters

- **modules** (*nn.ModuleDict*) – Dictionary of modules to be applied. Modules will be applied with ascending order of keys. We expect three types of modules: nn.Linear, nn.LayerNorm and MultiheadAttention.
- **encoded** (*torch.Tensor*) – Float Tensor that represents encoded vectors. Shape [batch_size, equation_length, hidden_size].
- **key_value** (*torch.Tensor*) – Float Tensor that represents key and value vectors when computing attention. Shape [batch_size, key_size, hidden_size].
- **key_ignorance_mask** (*torch.Tensor*) – Bool Tensor whose True values at (b, k) make attention layer ignore k-th key on b-th item in the batch. Shape [batch_size, key_size].
- **attention_mask** (*torch.BoolTensor*) – Bool Tensor whose True values at (t, k) make attention layer ignore k-th key when computing t-th query. Shape [equation_length, key_size].

Returns Float Tensor that indicates the scores under given information. Shape will be [batch_size, equation_length, ?]

Return type torch.Tensor

mwptoolkit.module.Decoder.ept_decoder.**get_embedding_without_pad**(*embedding: Union[torch.nn.modules.sparse.Embedding, torch.Tensor]*, *tokens: torch.Tensor*, *ignore_index=- 1*)

Get embedding vectors of given token tensor with ignored indices are zero-filled.

Parameters

- **embedding** (*nn.Embedding*) – An embedding instance
- **tokens** (*torch.Tensor*) – A Long Tensor to build embedding vectors.
- **ignore_index** (*int*) – Index to be ignored. *PAD_ID* by default.

Returns Embedding vector of given token tensor.

Return type torch.Tensor

mwptoolkit.module.Decoder.ept_decoder.**mask_forward**(*sz: int*, *diagonal: int = 1*)

Generate a mask that ignores future words. Each (i, j)-entry will be True if j >= i + diagonal

Parameters

- **sz** (*int*) – Length of the sequence.
- **diagonal** (*int*) – Amount of shift for diagonal entries.

Returns Mask tensor with shape [sz, sz].

Return type torch.Tensor

6.2.2 mwptoolkit.module.Decoder.rnn_decoder

class mwptoolkit.module.Decoder.rnn_decoder.**AttentionalRNNDecoder**(*embedding_size*, *hidden_size*, *context_size*, *num_dec_layers*, *rnn_cell_type*, *dropout_ratio=0.0*)

Bases: torch.nn.modules.module.Module

Attention-based Recurrent Neural Network (RNN) decoder.

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

forward(*input_embeddings*, *hidden_states=None*, *encoder_outputs=None*, *encoder_masks=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 output features, shape: [batch_size, sequence_length, num_directions * hidden_size]. hidden states, shape: [batch_size, num_layers * num_directions, hidden_size].

Return type tuple(torch.Tensor, torch.Tensor)

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 mwptoolkit.module.Decoder.rnn_decoder.**BasicRNNDecoder**(*embedding_size*, *hidden_size*, *num_layers*, *rnn_cell_type*, *dropout_ratio=0.0*)

Bases: torch.nn.modules.module.Module

Basic Recurrent Neural Network (RNN) decoder.

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

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

Return type tuple(torch.Tensor, torch.Tensor)

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 mwptoolkit.module.Decoder.rnn_decoder.**SalignedDecoder**(*operations*, *dim_hidden=300*, *dropout_rate=0.5*, *device=None*)

Bases: torch.nn.modules.module.Module

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

forward(*context*, *text_len*, *operands*, *stacks*, *prev_op*, *prev_output*, *prev_state*, *number_emb*, *N_OPS*)

Parameters

- **context** (*torch.Tensor*) – Encoded context, with size [batch_size, text_len, dim_hidden].
- **text_len** (*torch.Tensor*) – Text length for each problem in the batch.

- **operands** (*list of torch.Tensor*) – List of operands embeddings for each problem in the batch. Each element in the list is of size [n_operands, dim_hidden].
- **stacks** (*list of StackMachine*) – List of stack machines used for each problem.
- **prev_op** (*torch.LongTensor*) – Previous operation, with size [batch, 1].
- **prev_arg** (*torch.LongTensor*) – Previous argument indices, with size [batch, 1]. Can be None for the first step.
- **prev_output** (*torch.Tensor*) – Previous decoder RNN outputs, with size [batch, dim_hidden]. Can be None for the first step.
- **prev_state** (*torch.Tensor*) – Previous decoder RNN state, with size [batch, dim_hidden]. Can be None for the first step.

Returns op_logits: Logits of operation selection. arg_logits: Logits of argument choosing. outputs: Outputs of decoder RNN. state: Hidden state of decoder RNN.

Return type tuple(torch.Tensor, torch.Tensor, torch.Tensor, torch.Tensor)

pad_and_cat(*tensors*, *padding*)

Pad lists to have same number of elements, and concatenate those elements to a 3d tensor.

Parameters

- **tensors** (*list of list of Tensors*) – Each list contains list of operand embeddings. Each operand embedding is of size (dim_element,).
- **padding** (*Tensor*) – Element used to pad lists, with size (dim_element,).

Returns Length of lists in tensors. tensors (Tensor): Concatenated tensor after padding the list.

Return type n_tensors (list of int)

training: bool

6.2.3 mwptoolkit.module.Decoder.transformer_decoder

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

Bases: torch.nn.modules.module.Module

The stacked Transformer decoder layers.

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

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

6.2.4 mwptoolkit.module.Decoder.tree_decoder

class mwptoolkit.module.Decoder.tree_decoder.**HMSDecoder**(*embedding_model*, *hidden_size*, *dropout*, *op_set*, *vocab_dict*, *class_list*, *device*)

Bases: torch.nn.modules.module.Module

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

forward(*targets=None*, *encoder_hidden=None*, *encoder_outputs=None*, *input_lengths=None*, *span_length=None*, *num_pos=None*, *max_length=None*, *beam_width=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.

---

forward_beam(*decoder_init_hidden*, *encoder_outputs*, *masks*, *embedding_masks*, *max_length*, *beam_width=1*)

forward_step(*node_stacks*, *tree_stacks*, *nodes_hidden*, *encoder_outputs*, *masks*, *embedding_masks*, *decoder_nodes_class=None*)

forward_teacher(*decoder_nodes_label*, *decoder_init_hidden*, *encoder_outputs*, *masks*, *embedding_masks*, *max_length=None*)

get_class_embedding_mask(*num_pos*, *encoder_outputs*)

get_generator_embedding_mask(*batch_size*)

get_mask(*encode_lengths*, *pad_length*)

get_pad_masks(*encoder_outputs*, *input_lengths*, *span_length=None*)

get_pointer_embedding(*pointer_num_pos*, *encoder_outputs*)

get_pointer_mask(*pointer_num_pos*)

get_pointer_meta(*num_pos*, *sub_num_poses=None*)

get_predict_meta(*class_list*, *vocab_dict*, *device*)

init_stacks(*encoder_hidden*)

training: bool

class mwptoolkit.module.Decoder.tree_decoder.**LSTMBasedTreeDecoder**(*embedding_size*, *hidden_size*, *op_nums*, *generate_size*, *dropout=0.5*)

Bases: torch.nn.modules.module.Module

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

forward(*parent_embed*, *left_embed*, *prev_embed*, *encoder_outputs*, *num_pades*, *padding_hidden*, *seq_mask*, *nums_mask*, *hidden*, *tree_hidden*)

Parameters

- **parent_embed** (*list*) – parent embedding, length [batch_size], list of torch.Tensor with shape [1, 2 * hidden_size].
- **left_embed** (*list*) – left embedding, length [batch_size], list of torch.Tensor with shape [1, embedding_size].
- **prev_embed** (*list*) – previous embedding, length [batch_size], list of torch.Tensor with shape [1, embedding_size].
- **encoder_outputs** (*torch.Tensor*) – output from encoder, shape [batch_size, sequence_length, hidden_size].
- **num_pades** (*torch.Tensor*) – number representation, shape [batch_size, number_size, hidden_size].
- **padding_hidden** (*torch.Tensor*) – padding hidden, shape [1,hidden_size].
- **seq_mask** (*torch.BoolTensor*) – sequence mask, shape [batch_size, sequence_length].
- **mask_nums** (*torch.BoolTensor*) – number mask, shape [batch_size, number_size].
- **hidden** (*tuple(torch.Tensor, torch.Tensor)*) – hidden states, shape [batch_size, num_directions * hidden_size].
- **tree_hidden** (*tuple(torch.Tensor, torch.Tensor)*) – tree hidden states, shape [batch_size, num_directions * hidden_size].

Returns num_score, number score, shape [batch_size, number_size]. op, operator score, shape [batch_size, operator_size]. current_embeddings, current node representation, shape [batch_size, 1, num_directions * hidden_size]. current_context, current context representation, shape [batch_size, 1, num_directions * hidden_size]. embedding_weight, embedding weight, shape [batch_size, number_size, embedding_size]. hidden (tuple(torch.Tensor, torch.Tensor)): hidden states, shape [batch_size, num_directions * hidden_size]. tree_hidden (tuple(torch.Tensor, torch.Tensor)): tree hidden states, shape [batch_size, num_directions * hidden_size].

Return type tuple(torch.Tensor, torch.Tensor, torch.Tensor, torch.Tensor, torch.Tensor)

training: bool

class mwptoolkit.module.Decoder.tree_decoder.**PredictModel**(*hidden_size*, *class_size*, *dropout=0.4*)

Bases: torch.nn.modules.module.Module

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

forward(*node_hidden*, *encoder_outputs*, *masks*, *embedding_masks*)

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.

---

score_pn(*hidden*, *context*, *embedding_masks*)

training: bool

class mwptoolkit.module.Decoder.tree_decoder.**RNNBasedTreeDecoder**(*input_size*, *embedding_size*, *hidden_size*, *dropout_ratio*)

Bases: torch.nn.modules.module.Module

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

forward(*input_src*, *prev_c*, *prev_h*, *parent_h*, *sibling_state*)

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 mwptoolkit.module.Decoder.tree_decoder.**SARTreeDecoder**(*hidden_size*, *op_nums*, *generate_size*, *dropout=0.5*)

Bases: torch.nn.modules.module.Module

Seq2tree decoder with Semantically-Aligned Regularization

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

Semantically_Aligned_Regularization(*subtree_emb*, *s_aligned_vector*)

Parameters

- **subtree_emb** (*torch.Tensor*) –
- **s_aligned_vector** (*torch.Tensor*) –

Returns s_aligned_a s_aligned_d

Return type tuple(torch.Tensor, torch.Tensor)

forward(*node_stacks*, *left_childs*, *encoder_outputs*, *num_pades*, *padding_hidden*, *seq_mask*, *nums_mask*)

Parameters

- **node_stacks** (*list*) – node stacks.
- **left_childs** (*list*) – representation of left childs.
- **encoder_outputs** (*torch.Tensor*) – output from encoder, shape [sequence_length, batch_size, hidden_size].
- **num_pades** (*torch.Tensor*) – number representation, shape [batch_size, number_size, hidden_size].
- **padding_hidden** (*torch.Tensor*) – padding hidden, shape [1,hidden_size].
- **seq_mask** (*torch.BoolTensor*) – sequence mask, shape [batch_size, sequence_length].
- **mask_nums** (*torch.BoolTensor*) – number mask, shape [batch_size, number_size]

Returns num_score, number score, shape [batch_size, number_size]. op, operator score, shape [batch_size, operator_size]. current_node, current node representation, shape [batch_size, 1, hidden_size]. current_context, current context representation, shape [batch_size, 1, hidden_size]. embedding_weight, embedding weight, shape [batch_size, number_size, hidden_size].

Return type tuple(torch.Tensor, torch.Tensor, torch.Tensor, torch.Tensor, torch.Tensor)

training: bool

class mwptoolkit.module.Decoder.tree_decoder.**TreeDecoder**(*hidden_size*, *op_nums*, *generate_size*, *dropout=0.5*)

Bases: torch.nn.modules.module.Module

Seq2tree decoder with Problem aware dynamic encoding

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

forward(*node_stacks*, *left_childs*, *encoder_outputs*, *num_pades*, *padding_hidden*, *seq_mask*, *nums_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.

---

training: bool

6.3 mwptoolkit.module.Embedder

6.3.1 mwptoolkit.module.Embedder.basic_embedder

class mwptoolkit.module.Embedder.basic_embedder.**BasicEmbedder**(*input_size*, *embedding_size*, *dropout_ratio*, *padding_idx=0*)

Bases: torch.nn.modules.module.Module

Basic embedding layer

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

forward(*input_seq*)

Implement the embedding process :param input_seq: source sequence, shape [batch_size, sequence_length]. :type input_seq: torch.Tensor

Retruns: torch.Tensor: embedding output, shape [batch_size, sequence_length, embedding_size].

init_embedding_params(*sentences*, *vocab*)

training: bool

6.3.2 mwptoolkit.module.Embedder.bert_embedder

class mwptoolkit.module.Embedder.bert_embedder.**BertEmbedder**(*input_size*, *pretrained_model_path*)

Bases: torch.nn.modules.module.Module

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

forward(*input_seq*)

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.

---

token_resize(*input_size*)

training: bool

6.3.3 mwptoolkit.module.Embedder.position_embedder

class mwptoolkit.module.Embedder.position_embedder.**DisPositionalEncoding**(*embedding_size*, *max_len*)

Bases: torch.nn.modules.module.Module

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

forward(*dis_graph*, *category_num*)

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 mwptoolkit.module.Embedder.position_embedder.**EPTPositionalEncoding**(*embedding_dim*)

Bases: torch.nn.modules.module.Module

Positional encoding that extends trigonometric embedding proposed in 'Attention is all you need'

Instantiate positional encoding instance.

Parameters **embedding_dim** (*int*) – Dimension of embedding vector

_forward(*index_or_range*, *ignored_index=- 1*) → torch.Tensor

Compute positional encoding

$$P_{t,p} = c_p * \cos(a_p * t + b_p) + d_p * \sin(a_p * t + b_p).$$

Parameters

- **index_or_range** (*Union[torch.Tensor,int,range]*) – Value that represents positional encodings to be built. - A Tensor value indicates indices itself. - A integer value indicates indices from 0 to the value - A range value indicates indices within the range.
- **ignored_index** (*int*) – The index to be ignored. *PAD_ID* by default.

Return type torch.Tensor

Returns Positional encoding of given value. - If torch.Tensor of shape [*, L] is given, this will have shape [*, L, E] if L is not 1, otherwise [*, E]. - If integer or range is given, this will have shape [T, E], where T is the length of range.

before_trigonometric(*indices: torch.Tensor*) → torch.Tensor

Compute a_p * t + b_p for each index t. :param torch.Tensor indices: A Long tensor to compute indices. :rtype: torch.Tensor :return: Tensor whose values are a_p * t + b_p for each (t, p) entry.

property device: torch.device

Get the device where weights are currently put. :rtype: torch.device :return: Device instance

embedding_dim

Dimension of embedding vector

forward(*index_or_range*, *ignored_index=- 1*) → torch.Tensor

Compute positional encoding. If this encoding is not learnable, the result cannot have any gradient vector.

$$P_{t,p} = c_p * \cos(a_p * t + b_p) + d_p * \sin(a_p * t + b_p).$$

Parameters

- **index_or_range** (*Union[torch.Tensor,int,range]*) – Value that represents positional encodings to be built. - A Tensor value indicates indices itself. - A integer value indicates indices from 0 to the value - A range value indicates indices within the range.
- **ignored_index** (*int*) – The index to be ignored. *PAD_ID* by default.

Return type torch.Tensor

Returns Positional encoding of given value. - If torch.Tensor of shape [*, L] is given, this will have shape [*, L, E] if L is not 1, otherwise [*, E]. - If integer or range is given, this will have shape [T, E], where T is the length of range.

training: bool

class mwptoolkit.module.Embedder.position_embedder.**PositionEmbedder**(*embedding_size*, *max_length=512*)

Bases: torch.nn.modules.module.Module

This module produces sinusoidal positional embeddings of any length.

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

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

Parameters **input_seq** (*torch.Tensor*) – input sequence, shape [batch_size, sequence_length].

Returns position embedding, shape [batch_size, sequence_length, embedding_size].

Return type torch.Tensor

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

class mwptoolkit.module.Embedder.position_embedder.**PositionEmbedder_x**(*embedding_size*, *max_len=1024*)

Bases: torch.nn.modules.module.Module

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

forward(*input_embedding*)

Parameters **input_embedding** (*torch.Tensor*) – shape [batch_size, sequence_length, embedding_size].

training: bool

class mwptoolkit.module.Embedder.position_embedder.**PositionalEncoding**(*pos_size*, *dim*)

Bases: torch.nn.modules.module.Module

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

forward(*input*)

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

6.3.4 mwptoolkit.module.Embedder.roberta_embedder

class mwptoolkit.module.Embedder.roberta_embedder.**RobertaEmbedder**(*input_size*, *pretrained_model_path*)

Bases: torch.nn.modules.module.Module

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

forward(*input_seq*, *attn_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.

---

token_resize(*input_size*)

training: bool

6.4 mwptoolkit.module.Encoder

6.4.1 mwptoolkit.module.Encoder.graph_based_encoder

class mwptoolkit.module.Encoder.graph_based_encoder.**GraphBasedEncoder**(*embedding_size*, *hidden_size*, *rnn_cell_type*, *bidirectional*, *num_layers=2*, *dropout_ratio=0.5*)

Bases: torch.nn.modules.module.Module

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

forward(*input_embedding*, *input_lengths*, *batch_graph*, *hidden=None*)

Parameters

- **input_embedding** (*torch.Tensor*) – input variable, shape [sequence_length, batch_size, embedding_size].
- **input_lengths** (*torch.Tensor*) – length of input sequence, shape: [batch_size].
- **batch_graph** (*torch.Tensor*) – graph input variable, shape [batch_size, 5, sequence_length, sequence_length].

Returns pade_outputs, encoded variable, shape [sequence_length, batch_size, hidden_size]. problem_output, vector representation of problem, shape [batch_size, hidden_size].

Return type tuple(torch.Tensor, torch.Tensor)

training: bool

class mwptoolkit.module.Encoder.graph_based_encoder.**GraphBasedMultiEncoder**(*input1_size*, *input2_size*, *embed_model*, *embedding1_size*, *embedding2_size*, *hidden_size*, *n_layers=2*, *hop_size=2*, *dropout=0.5*)

Bases: torch.nn.modules.module.Module

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

forward(*input1_var*, *input2_var*, *input_length*, *parse_graph*, *hidden=None*)

training: bool

class mwptoolkit.module.Encoder.graph_based_encoder.**GraphEncoder**(*vocab_size*, *embedding_size*, *hidden_size*, *sample_size*, *sample_layer*, *bidirectional*, *dropout_ratio*)

Bases: torch.nn.modules.module.Module

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

forward(*fw_adj_info*, *bw_adj_info*, *feature_info*, *batch_nodes*)

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 mwptoolkit.module.Encoder.graph_based_encoder.**NumEncoder**(*node_dim*, *hop_size=2*)

Bases: torch.nn.modules.module.Module

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

forward(*encoder_outputs*, *num_encoder_outputs*, *num_pos_pad*, *num_order_pad*)

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

mwptoolkit.module.Encoder.graph_based_encoder.**replace_masked_values**(*tensor*, *mask*, *replace_with*)

6.4.2 mwptoolkit.module.Encoder.rnn_encoder

class mwptoolkit.module.Encoder.rnn_encoder.**BasicRNNEncoder**(*embedding_size*, *hidden_size*, *num_layers*, *rnn_cell_type*, *dropout_ratio*, *bidirectional=True*, *batch_first=True*)

Bases: torch.nn.modules.module.Module

Basic Recurrent Neural Network (RNN) encoder.

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

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

Return type tuple(torch.Tensor, torch.Tensor)

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 mwptoolkit.module.Encoder.rnn_encoder.**GroupAttentionRNNEncoder**(*emb_size=100*, *hidden_size=128*, *n_layers=1*, *bidirectional=False*, *rnn_cell=None*, *rnn_cell_name='gru'*, *variable_lengths=True*, *d_ff=2048*, *dropout=0.3*, *N=1*)

Bases: torch.nn.modules.module.Module

Group Attentional Recurrent Neural Network (RNN) encoder.

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

forward(*embedded*, *input_var*, *split_list*, *input_lengths=None*)

Parameters

- **embedded** (*torch.Tensor*) – embedded inputs, shape [batch_size, sequence_length, embedding_size].

- **input_var** (*torch.Tensor*) – source sequence, shape [batch_size, sequence_length].
- **split_list** (*list*) – group split index.
- **input_lengths** (*torch.Tensor*) – length of input sequence, shape: [batch_size].

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

Return type tuple(torch.Tensor, torch.Tensor)

training: bool

class mwptoolkit.module.Encoder.rnn_encoder.**HWCPEncoder**(*embedding_model*, *embedding_size*, *hidden_size=512*, *span_size=0*, *dropout_ratio=0.4*)

Bases: torch.nn.modules.module.Module

Hierarchical word-clause-problem encoder

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

bi_combine(*output*, *hidden*)

clause_level_forward(*word_output*, *tree_batch*)

dependency_encode(*word_output*, *node*)

forward(*input_var*, *input_lengths*, *span_length*, *tree=None*, *output_all_layers=False*)

Not implemented

get_mask(*encode_lengths*, *pad_length*)

problem_level_forword(*span_input*, *span_mask*)

training: bool

word_level_forward(*embedding_inputs*, *input_length*, *bi_word_hidden=None*)

class mwptoolkit.module.Encoder.rnn_encoder.**SalignedEncoder**(*dim_embed*, *dim_hidden*, *dim_last*, *dropout_rate*, *dim_attn_hidden=256*)

Bases: torch.nn.modules.module.Module

Simple RNN encoder with attention which also extract variable embedding.

Parameters

- **dim_embed** (*int*) – Dimension of input embedding.
- **dim_hidden** (*int*) – Dimension of encoder RNN.
- **dim_last** (*int*) – Dimension of the last state will be transformed to.
- **dropout_rate** (*float*) – Dropout rate.

forward(*inputs*, *lengths*, *constant_indices*)

Parameters

- **inputs** (*torch.Tensor*) – Indices of words, shape [batch_size, sequence_length].
- **length** (*torch.Tensor*) – Length of inputs, shape [batch_size].
- **constant_indices** (*list of int*) – Each list contains list.

Returns Encoded sequence, shape [batch_size, sequence_length, hidden_size].

Return type torch.Tensor

get_fix_constant()

initialize_fix_constant(*con_len*, *device*)

training: bool

class mwptoolkit.module.Encoder.rnn_encoder.**SelfAttentionRNNEncoder**(*embedding_size*, *hidden_size*, *context_size*, *num_layers*, *rnn_cell_type*, *dropout_ratio*, *bidirectional=True*)

Bases: torch.nn.modules.module.Module

Self Attentional Recurrent Neural Network (RNN) encoder.

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

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

Return type tuple(torch.Tensor, torch.Tensor)

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

6.4.3 mwptoolkit.module.Encoder.transformer_encoder

class mwptoolkit.module.Encoder.transformer_encoder.**BertEncoder**(*hidden_size*, *dropout_ratio*, *pretrained_model_path*)

Bases: torch.nn.modules.module.Module

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

forward(*input_ids*, *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.

---

token_resize(*input_size*)

training: bool

class mwptoolkit.module.Encoder.transformer_encoder.**GroupATTEncoder**(*layer*, *N*)

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

Group attentional encoder, N layers of group attentional encoder layer.

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

forward(*inputs*, *mask*)

Pass the input (and mask) through each layer in turn.

Parameters **inputs** (*torch.Tensor*) – input variavle, shape [batch_size, sequence_length, hidden_size].

Returns encoded variavle, shape [batch_size, sequence_length, hidden_size].

Return type torch.Tensor

training: bool

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

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

The stacked Transformer encoder layers.

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

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`

6.5 mwptoolkit.module.Environment

6.5.1 mwptoolkit.module.Environment.stack_machine

`class mwptoolkit.module.Environment.stack_machine.OPERATIONS(out_symbol2idx)`

Bases: `object`

`class mwptoolkit.module.Environment.stack_machine.StackMachine(operations, constants, embeddings, bottom_embedding, dry_run=False)`

Bases: `object`

Parameters

- **constants** (*list*) – Value of numbers.
- **embeddings** (*tensor*) – Tensor of shape [len(constants), dim_embedding]. Embedding of the constants.
- **bottom_embedding** (*teonsor*) – Tensor of shape (dim_embedding,). The embeding to return when stack is empty.

`add_variable(embedding)`

Tell the stack machine to increase the number of nuknown variables by 1.

Parameters **embedding** (*torch.Tensor*) – Tensor of shape (dim_embedding). Embedding of the unknown varialbe.

`apply_embed_only(operation, embed_res)`

Apply operator on stack with embedding operation only.

Parameters

- **operator** (*mwptoolkit.module.Environment.stack_machine.OPERATION*) – One of - OPERATIONS.ADD - OPERATIONS.SUB - OPERATIONS.MUL - OPERATIONS.DIV - OPERATIONS.EQL
- **embed_res** (*torch.FloatTensor*) – Resulted embedding after transformation, with size (dim_embedding,).

Returns embedding on the top of the stack.

Return type torch.Tensor

`apply_eql(operation)`

`get_height()`

Get the height of the stack.

Returns height.

Return type int

get_solution()

Get solution. If the problem has not been solved, return None.

Returns If the problem has been solved, return result from sympy.solve. If not, return None.

Return type list

get_stack()

get_top2()

Get the top 2 embeddings of the stack.

Returns Return tensor of shape (2, embed_dim).

Return type torch.Tensor

push(*operand_index*)

Push var to stack.

Parameters **operand_index** (*int*) – Index of the operand. If index >= number of constants, then it implies a variable is pushed.

Returns Simply return the pushed embedding.

Return type torch.Tensor

6.6 mwptoolkit.module.Graph

6.6.1 mwptoolkit.module.Graph.gcn

class mwptoolkit.module.Graph.gcn.**GCN**(*in_feat_dim*, *nhid*, *out_feat_dim*, *dropout*)

Bases: torch.nn.modules.module.Module

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

forward(*x*, *adj*)

Parameters

- **x** (*torch.Tensor*) – input features, shape [batch_size, node_num, in_feat_dim]
- **adj** (*torch.Tensor*) – adjacency matrix, shape [batch_size, node_num, node_num]

Returns gcn_enhance_feature, shape [batch_size, node_num, out_feat_dim]

Return type torch.Tensor

training: bool

6.6.2 mwptoolkit.module.Graph.graph_module

class mwptoolkit.module.Graph.graph_module.**Graph_Module**(*indim*, *hiddim*, *outdim*, *dropout=0.3*)

Bases: torch.nn.modules.module.Module

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

b_normal(*adj*)

forward(*graph_nodes*, *graph*)

Parameters **graph_nodes** (*torch.Tensor*) – input features, shape [batch_size, node_num, in_feat_dim]

Returns graph_encode_features, shape [batch_size, node_num, out_feat_dim]

Return type torch.Tensor

get_adj(*graph_nodes*)

Parameters **graph_nodes** (*torch.Tensor*) – input features, shape [batch_size, node_num, in_feat_dim]

Returns adjacency matrix, shape [batch_size, node_num, node_num]

Return type torch.Tensor

normalize(*A*, *symmetric=True*)

Parameters **A** (*torch.Tensor*) – adjacency matrix (node_num, node_num)

Returns adjacency matrix (node_num, node_num)

training: bool

class mwptoolkit.module.Graph.graph_module.**Num_Graph_Module**(*node_dim*)

Bases: torch.nn.modules.module.Module

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

forward(*node*, *graph1*, *graph2*)

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.

---

normalize(*graph*, *symmetric=True*)

training: bool

class mwptoolkit.module.Graph.graph_module.**Parse_Graph_Module**(*hidden_size*)

Bases: torch.nn.modules.module.Module

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

forward(*node*, *graph*)

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.

---

normalize(*graph*, *symmetric=True*)

training: bool

6.7 mwptoolkit.module.Layer

6.7.1 mwptoolkit.module.Layer.graph_layers

class mwptoolkit.module.Layer.graph_layers.**GraphConvolution**(*in_features*, *out_features*, *bias=True*)

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

Simple GCN layer, similar to https://arxiv.org/abs/1609.02907

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

forward(*input*, *adj*)

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.

---

reset_parameters()

training: bool

class mwptoolkit.module.Layer.graph_layers.**LayerNorm**(*features*, *eps=1e-06*)

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

Construct a layernorm module (See citation for details).

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

forward(*x*)

Parameters **x** (*torch.Tensor*) – input variable.

Returns output variable.

Return type torch.Tensor

training: bool

class mwptoolkit.module.Layer.graph_layers.**MeanAggregator**(*input_dim*, *output_dim*, *activation=<function relu>*, *concat=False*)

Bases: torch.nn.modules.module.Module

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

forward(*inputs*)

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 mwptoolkit.module.Layer.graph_layers.**PositionwiseFeedForward**(*d_model*, *d_ff*, *d_out*, *dropout=0.1*)

Bases: torch.nn.modules.module.Module

Implements FFN equation.

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

forward(*x*)

Parameters **x** (*torch.Tensor*) – input variable.

Returns output variable.

Return type torch.Tensor

training: bool

6.7.2 mwptoolkit.module.Layer.layers

class mwptoolkit.module.Layer.layers.**GenVar**(*dim_encoder_state*, *dim_context*, *dim_attn_hidden=256*, *dropout_rate=0.5*)

Bases: torch.nn.modules.module.Module

Module to generate variable embedding.

Parameters

- **dim_encoder_state** (*int*) – Dimension of the last cell state of encoder RNN (output of Encoder module).
- **dim_context** (*int*) – Dimension of RNN in GenVar module.
- **dim_attn_hidden** (*int*) – Dimension of hidden layer in attention.
- **dim_mlp_hiddens** (*int*) – Dimension of hidden layers in the MLP that transform encoder state to query of attention.
- **dropout_rate** (*int*) – Dropout rate for attention and MLP.

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

forward(*encoder_state*, *context*, *context_lens*)

Generate embedding for an unknown variable.

Parameters

- **encoder_state** (*torch.FloatTensor*) – Last cell state of the encoder (output of Encoder module).
- **context** (*torch.FloatTensor*) – Encoded context, with size [batch_size, text_len, dim_hidden].

Returns Embedding of an unknown variable, with size [batch_size, dim_context]

Return type torch.FloatTensor

training: bool

class mwptoolkit.module.Layer.layers.**Transformer**(*dim_hidden*)

Bases: torch.nn.modules.module.Module

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

forward(*top2*)

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 mwptoolkit.module.Layer.layers.**TreeAttnDecoderRNN**(*hidden_size*, *embedding_size*, *input_size*, *output_size*, *n_layers=2*, *dropout=0.5*)

Bases: torch.nn.modules.module.Module

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

forward(*input_seq*, *last_hidden*, *encoder_outputs*, *seq_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.

---

training: bool

6.7.3 mwptoolkit.module.Layer.transformer_layer

class mwptoolkit.module.Layer.transformer_layer.**EPTTransformerLayer**(*hidden_dim=None*, *num_decoder_heads=None*, *layernorm_eps=None*, *intermediate_dim=None*)

Bases: torch.nn.modules.module.Module

Class for Transformer Encoder/Decoder layer (follows the paper, 'Attention is all you need')

Initialize TransformerLayer class

Parameters **config** (*ModelConfig*) – Configuration of this Encoder/Decoder layer

forward(*target*, *target_ignorance_mask=None*, *target_attention_mask=None*, *memory=None*, *memory_ignorance_mask=None*)

Forward-computation of Transformer Encoder/Decoder layers

Parameters

- **target** (*torch.Tensor*) – FloatTensor indicating Sequence of target vectors. Shape [batch_size, target_length, hidden_size].
- **target_ignorance_mask** (*torch.Tensor*) – BoolTensor indicating Mask for target tokens that should be ignored. Shape [batch_size, target_length].
- **target_attention_mask** (*torch.Tensor*) – BoolTensor indicating Target-to-target Attention mask for target tokens. Shape [target_length, target_length].
- **memory** (*torch.Tensor*) – FloatTensor indicating Sequence of source vectors. Shape [batch_size, sequence_length, hidden_size]. This can be None when you want to use this layer as an encoder layer.
- **memory_ignorance_mask** (*torch.Tensor*) – BoolTensor indicating Mask for source tokens that should be ignored. Shape [batch_size, sequence_length].

Returns Decoder hidden states per each target token, shape [batch_size, sequence_length, hidden_size].

Return type torch.FloatTensor

training: **bool**

class mwptoolkit.module.Layer.transformer_layer.**GAEncoderLayer**(*size*, *self_attn*, *feed_forward*, *dropout*)

Bases: torch.nn.modules.module.Module

Group attentional encoder layer, encoder is made up of self-attn and feed forward.

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

forward(*x*, *mask*)

Follow Figure 1 (left) for connections.

training: **bool**

class mwptoolkit.module.Layer.transformer_layer.**LayerNorm**(*features*, *eps=1e-06*)

Bases: torch.nn.modules.module.Module

Construct a layernorm module (See citation for details).

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.

---

training: bool

class mwptoolkit.module.Layer.transformer_layer.**PositionwiseFeedForward**(*d_model*, *d_ff*, *dropout=0.1*)

Bases: torch.nn.modules.module.Module

Implements FFN equation.

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.

---

training: bool

class mwptoolkit.module.Layer.transformer_layer.**SublayerConnection**(*size*, *dropout*)

Bases: torch.nn.modules.module.Module

A residual connection followed by a layer norm. Note for code simplicity the norm is first as opposed to last.

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

forward(*x*, *sublayer*)

Apply residual connection to any sublayer with the same size.

training: bool

class mwptoolkit.module.Layer.transformer_layer.**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: torch.nn.modules.module.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)

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

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

6.7.4 mwptoolkit.module.Layer.tree_layers

class mwptoolkit.module.Layer.tree_layers.**DQN**(*input_size*, *embedding_size*, *hidden_size*, *output_size*, *dropout_ratio*)

Bases: torch.nn.modules.module.Module

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

forward(*inputs*)

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.

play_one(*inputs*)

training: bool

class mwptoolkit.module.Layer.tree_layers.**Dec_LSTM**(*embedding_size*, *hidden_size*, *dropout_ratio*)

Bases: torch.nn.modules.module.Module

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

forward(*x*, *prev_c*, *prev_h*, *parent_h*, *sibling_state*)

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 mwptoolkit.module.Layer.tree_layers.**DecomposeModel**(*hidden_size*, *dropout*, *device*)

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

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

forward(*node_stacks*, *tree_stacks*, *nodes_context*, *labels_embedding*, *pad_node=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.

---

training: bool

class mwptoolkit.module.Layer.tree_layers.**GateNN**(*hidden_size*, *input1_size*, *input2_size=0*, *dropout=0.4*, *single_layer=False*)

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

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

forward(*hidden*, *input1*, *input2=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

class mwptoolkit.module.Layer.tree_layers.**GenerateNode**(*hidden_size*, *op_nums*, *embedding_size*, *dropout=0.5*)

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

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

forward(*node_embedding*, *node_label*, *current_context*)

Parameters

- **node_embedding** (*torch.Tensor*) – node embedding, shape [batch_size, hidden_size].

- **node_label** (*torch.Tensor*) – representation of node label, shape [batch_size, embedding_size].
- **current_context** (*torch.Tensor*) – current context, shape [batch_size, hidden_size].

Returns l_child, representation of left child, shape [batch_size, hidden_size]. r_child, representation of right child, shape [batch_size, hidden_size]. **node_label_**, representation of node label, shape [batch_size, embedding_size].

Return type tuple(torch.Tensor, torch.Tensor, torch.Tensor)

training: bool

class mwptoolkit.module.Layer.tree_layers.**Merge**(*hidden_size*, *embedding_size*, *dropout=0.5*)

Bases: torch.nn.modules.module.Module

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

forward(*node_embedding*, *sub_tree_1*, *sub_tree_2*)

Parameters

- **node_embedding** (*torch.Tensor*) – node embedding, shape [1, embedding_size].
- **sub_tree_1** (*torch.Tensor*) – representation of sub tree 1, shape [1, hidden_size].
- **sub_tree_2** (*torch.Tensor*) – representation of sub tree 2, shape [1, hidden_size].

Returns representation of merged tree, shape [1, hidden_size].

Return type torch.Tensor

training: bool

class mwptoolkit.module.Layer.tree_layers.**Node**(*node_value*, *isleaf=True*)

Bases: object

set_left_node(*node*)

set_right_node(*node*)

class mwptoolkit.module.Layer.tree_layers.**NodeEmbeddingLayer**(*op_nums*, *embedding_size*)

Bases: torch.nn.modules.module.Module

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

forward(*node_embedding*, *node_label*, *current_context*)

Parameters

- **node_embedding** (*torch.Tensor*) – node embedding, shape [batch_size, num_directions * hidden_size].
- **node_label** (*torch.Tensor*) – shape [batch_size].

Returns l_child, representation of left child, shape [batch_size, num_directions * hidden_size]. r_child, representation of right child, shape [batch_size, num_directions * hidden_size]. **node_label_**, representation of node label, shape [batch_size, embedding_size].

Return type tuple(torch.Tensor, torch.Tensor, torch.Tensor)

training: bool

class mwptoolkit.module.Layer.tree_layers.**NodeEmbeddingNode**(*node_hidden*, *node_context=None*, *label_embedding=None*)

Bases: object

class mwptoolkit.module.Layer.tree_layers.**NodeGenerater**(*hidden_size*, *op_nums*, *embedding_size*, *dropout=0.5*)

Bases: torch.nn.modules.module.Module

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

forward(*node_embedding*, *node_label*, *current_context*)

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 mwptoolkit.module.Layer.tree_layers.**Prediction**(*hidden_size*, *op_nums*, *input_size*, *dropout=0.5*)

Bases: torch.nn.modules.module.Module

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

forward(*node_stacks*, *left_childs*, *encoder_outputs*, *num_pades*, *padding_hidden*, *seq_mask*, *mask_nums*)

Parameters

- **node_stacks** (*list*) – node stacks.
- **left_childs** (*list*) – representation of left childs.
- **encoder_outputs** (*torch.Tensor*) – output from encoder, shape [sequence_length, batch_size, hidden_size].
- **num_pades** (*torch.Tensor*) – number representation, shape [batch_size, number_size, hidden_size].
- **padding_hidden** (*torch.Tensor*) – padding hidden, shape [1,hidden_size].
- **seq_mask** (*torch.BoolTensor*) – sequence mask, shape [batch_size, sequence_length].
- **mask_nums** (*torch.BoolTensor*) – number mask, shape [batch_size, number_size].

Returns num_score, number score, shape [batch_size, number_size]. op, operator score, shape [batch_size, operator_size]. current_node, current node representation, shape [batch_size, 1, hidden_size]. current_context, current context representation, shape [batch_size, 1, hidden_size]. embedding_weight, embedding weight, shape [batch_size, number_size, hidden_size].

Return type tuple(torch.Tensor, torch.Tensor, torch.Tensor, torch.Tensor, torch.Tensor)

training: bool

class mwptoolkit.module.Layer.tree_layers.**RecursiveNN**(*emb_size*, *op_size*, *op_list*)

Bases: torch.nn.modules.module.Module

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

RecurCell(*combine_emb*)

forward(*expression_tree*, *num_embedding*, *look_up*, *out_idx2symbol*)

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.

---

leaf_emb(*node*, *num_embed*, *look_up*)

test(*expression_tree*, *num_embedding*, *look_up*, *out_idx2symbol*)

test_traverse(*node*)

training: bool

traverse(*node*)

class mwptoolkit.module.Layer.tree_layers.**Score**(*input_size*, *hidden_size*)

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

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

forward(*hidden*, *num_embeddings*, *num_mask=None*)

Parameters

- **hidden** (*torch.Tensor*) – hidden representation, shape [batch_size, 1, hidden_size + input_size].
- **num_embeddings** (*torch.Tensor*) – number embedding, shape [batch_size, number_size, hidden_size].
- **num_mask** (*torch.BoolTensor*) – number mask, shape [batch_size, number_size].

Returns shape [batch_size, number_size].

Return type score (torch.Tensor)

training: bool

class mwptoolkit.module.Layer.tree_layers.**ScoreModel**(*hidden_size*)

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

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

forward(*hidden*, *context*, *token_embeddings*)

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 mwptoolkit.module.Layer.tree_layers.**SemanticAlignmentModule**(*encoder_hidden_size*, *decoder_hidden_size*, *hidden_size*, *batch_first=False*)

Bases: torch.nn.modules.module.Module

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

forward(*decoder_hidden*, *encoder_outputs*)

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 mwptoolkit.module.Layer.tree_layers.**SubTreeMerger**(*hidden_size*, *embedding_size*, *dropout=0.5*)

Bases: torch.nn.modules.module.Module

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

forward(*node_embedding*, *sub_tree_1*, *sub_tree_2*)

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 mwptoolkit.module.Layer.tree_layers.**TreeAttention**(*input_size*, *hidden_size*)

Bases: torch.nn.modules.module.Module

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

forward(*hidden*, *encoder_outputs*, *seq_mask=None*)

Parameters

- **hidden** (*torch.Tensor*) – hidden representation, shape [1, batch_size, hidden_size]
- **encoder_outputs** (*torch.Tensor*) – output from encoder, shape [sequence_length, batch_size, hidden_size].
- **seq_mask** (*torch.Tensor*) – sequence mask, shape [batch_size, sequence_length].

Returns attention energies, shape [batch_size, 1, sequence_length].

Return type attn_energies (torch.Tensor)

training: bool

class mwptoolkit.module.Layer.tree_layers.**TreeEmbedding**(*embedding*, *terminal=False*)

Bases: object

class mwptoolkit.module.Layer.tree_layers.**TreeEmbeddingModel**(*hidden_size*, *op_set*, *dropout=0.4*)

Bases: torch.nn.modules.module.Module

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

forward(*class_embedding*, *tree_stacks*, *embed_node_index*)

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.

---

merge(*op_embedding*, *left_embedding*, *right_embedding*)

training: bool

class mwptoolkit.module.Layer.tree_layers.**TreeNode**(*embedding*, *left_flag=False*, *terminal=False*)

Bases: object

6.8 mwptoolkit.module.Strategy

6.8.1 mwptoolkit.module.Strategy.beam_search

class mwptoolkit.module.Strategy.beam_search.**Beam**(*score*, *input_var*, *hidden*, *token_logits*, *outputs*, *all_output=None*)

Bases: object

class mwptoolkit.module.Strategy.beam_search.**BeamNode**(*score*, *nodes_hidden*, *node_stacks*, *tree_stacks*, *decoder_outputs_list*, *sequence_symbols_list*)

Bases: object

copy()

class mwptoolkit.module.Strategy.beam_search.**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 mwptoolkit.module.Strategy.beam_search.**TreeBeam**(*score*, *node_stack*, *embedding_stack*, *left_childs*, *out*, *token_logit=None*)

Bases: `object`

6.8.2 mwptoolkit.module.Strategy.greedy

mwptoolkit.module.Strategy.greedy.**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

6.8.3 mwptoolkit.module.Strategy.sampling

mwptoolkit.module.Strategy.sampling.**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

CHAPTER

SEVEN

MWPTOOLKIT.TRAINER

7.1 mwptoolkit.trainer.abstract_trainer

class mwptoolkit.trainer.abstract_trainer.**AbstractTrainer**(*config*, *model*, *dataloader*, *evaluator*)

Bases: object

abstract trainer

the base class of trainer class.

example of instantiation:

```
>>> trainer = AbstractTrainer(config, model, dataloader, evaluator)
```

for training:

```
>>> trainer.fit()
```

for testing:

```
>>> trainer.test()
```

for parameter searching:

```
>>> trainer.param_search()
```

Parameters

- **config** (*config*) – An instance object of Config, used to record parameter information.
- **model** (*Model*) – An object of deep-learning model.
- **dataloader** (*Dataloader*) – dataloader object.
- **evaluator** (*Evaluator*) – evaluator object.

expected that config includes these parameters below:

test_step (int): the epoch number of training after which conducts the evaluation on test.

best_folds_accuracy (list|None): when running k-fold cross validation, this keeps the accuracy of folds that already run.

evaluate(*eval_set*)

fit()

param_search()

test()

7.2 mwptoolkit.trainer.supervised_trainer

class mwptoolkit.trainer.supervised_trainer.**BertTDTrainer**(*config*, *model*, *dataloader*, *evaluator*)

Bases: *mwptoolkit.trainer.supervised_trainer.SupervisedTrainer*

Parameters

- **config** (*config*) – An instance object of Config, used to record parameter information.
- **model** (*Model*) – An object of deep-learning model.
- **dataloader** (*Dataloader*) – dataloader object.
- **evaluator** (*Evaluator*) – evaluator object.

expected that config includes these parameters below:

learning_rate (float): learning rate of model

train_batch_size (int): the training batch size.

epoch_nums (int): number of epochs.

trained_model_path (str): a path of file which is used to save parameters of best model.

checkpoint_path (str): a path of file which is used save checkpoint of training progress.

output_path (str|None): a path of a json file which is used to save test output infomation fo model.

resume (bool): start training from last checkpoint.

validset_divide (bool): whether to split validset. if True, the dataset is split to trainset-validset-testset. if False, the dataset is split to trainset-testset.

test_step (int): the epoch number of training after which conducts the evaluation on test.

best_folds_accuracy (list|None): when running k-fold cross validation, this keeps the accuracy of folds that already run.

class mwptoolkit.trainer.supervised_trainer.**EPTTrainer**(*config*, *model*, *dataloader*, *evaluator*)

Bases: *mwptoolkit.trainer.abstract_trainer.AbstractTrainer*

ept trainer, used to implement training, testing, parameter searching for deep-learning model EPT.

example of instantiation:

```
>>> trainer = EPTTrainer(config, model, dataloader, evaluator)
```

for training:

```
>>> trainer.fit()
```

for testing:

```
>>> trainer.test()
```

for parameter searching:

```
>>> trainer.param_search()
```

Parameters

- **config** (*config*) – An instance object of Config, used to record parameter information.
- **model** (*Model*) – An object of deep-learning model.
- **dataloader** (*Dataloader*) – dataloader object.
- **evaluator** (*Evaluator*) – evaluator object.

expected that config includes these parameters below:

learning_rate (float): learning rate of model

train_batch_size (int): the training batch size.

epoch_nums (int): number of epochs.

gradient_accumulation_steps (int): gradient accumulation steps.

epoch_warmup (int): epoch warmup.

fix_encoder_embedding (bool): whether require gradient of embedding module of encoder

trained_model_path (str): a path of file which is used to save parameters of best model.

checkpoint_path (str): a path of file which is used save checkpoint of training progress.

output_path (str|None): a path of a json file which is used to save test output infomation fo model.

resume (bool): start training from last checkpoint.

validset_divide (bool): whether to split validset. if True, the dataset is split to trainset-validset-testset. if False, the dataset is split to trainset-testset.

test_step (int): the epoch number of training after which conducts the evaluation on test.

best_folds_accuracy (list|None): when running k-fold cross validation, this keeps the accuracy of folds that already run.

_eval_batch(*batch*)

seq, seq_length, group_nums, target

_normalize_gradients(**parameters*)

Normalize gradients (as in NVLAMB optimizer)

Parameters **parameters** – List of parameters whose gradient will be normalized.

Returns Frobenious Norm before applying normalization.

evaluate(*eval_set*)

evaluate model.

Parameters **eval_set** (*str*) – [valid | test], the dataset for evaluation.

Returns equation accuracy, value accuracy, count of evaluated datas, formatted time string of evaluation time.

Return type tuple(float,float,int,str)

fit()

train model.

param_search()

hyper-parameter search.

test()

test model.

class mwptoolkit.trainer.supervised_trainer.**GTSTrainer**(*config*, *model*, *dataloader*, *evaluator*)

Bases: *mwptoolkit.trainer.abstract_trainer.AbstractTrainer*

gts trainer, used to implement training, testing, parameter searching for deep-learning model GTS.

example of instantiation:

```
>>> trainer = GTSTrainer(config, model, dataloader, evaluator)
```

for training:

```
>>> trainer.fit()
```

for testing:

```
>>> trainer.test()
```

for parameter searching:

```
>>> trainer.param_search()
```

Parameters

- **config** (*config*) – An instance object of Config, used to record parameter information.
- **model** (*Model*) – An object of deep-learning model.
- **dataloader** (*Dataloader*) – dataloader object.
- **evaluator** (*Evaluator*) – evaluator object.

expected that config includes these parameters below:

learning_rate (float): learning rate of model.

embedding_learning_rate (float): learning rate of embedding module.

train_batch_size (int): the training batch size.

step_size (int): step_size of scheduler.

epoch_nums (int): number of epochs.

trained_model_path (str): a path of file which is used to save parameters of best model.

checkpoint_path (str): a path of file which is used save checkpoint of training progress.

output_path (str|None): a path of a json file which is used to save test output infomation fo model.

resume (bool): start training from last checkpoint.

validset_divide (bool): whether to split validset. if True, the dataset is split to trainset-validset-testset. if False, the dataset is split to trainset-testset.

test_step (int): the epoch number of training after which conducts the evaluation on test.

best_folds_accuracy (list|None): when running k-fold cross validation, this keeps the accuracy of folds that already run.

evaluate(*eval_set*)

evaluate model.

Parameters **eval_set** (*str*) – [valid | test], the dataset for evaluation.

Returns equation accuracy, value accuracy, count of evaluated datas, formatted time string of evaluation time.

Return type tuple(float,float,int,str)

fit()

train model.

param_search()

hyper-parameter search.

test()

test model.

class mwptoolkit.trainer.supervised_trainer.**Graph2TreeTrainer**(*config*, *model*, *dataloader*, *evaluator*)

Bases: *mwptoolkit.trainer.supervised_trainer.GTSTrainer*

graph2tree trainer, used to implement training, testing, parameter searching for deep-learning model Graph2Tree.

example of instantiation:

```
>>> trainer = Graph2TreeTrainer(config, model, dataloader, evaluator)
```

for training:

```
>>> trainer.fit()
```

for testing:

```
>>> trainer.test()
```

for parameter searching:

```
>>> trainer.param_search()
```

Parameters

- **config** (*config*) – An instance object of Config, used to record parameter information.
- **model** (*Model*) – An object of deep-learning model.
- **dataloader** (*Dataloader*) – dataloader object.
- **evaluator** (*Evaluator*) – evaluator object.

expected that config includes these parameters below:

learning_rate (float): learning rate of model.

embedding_learning_rate (float): learning rate of embedding module.

train_batch_size (int): the training batch size.

step_size (int): step_size of scheduler.

epoch_nums (int): number of epochs.

trained_model_path (str): a path of file which is used to save parameters of best model.

checkpoint_path (str): a path of file which is used save checkpoint of training progress.

output_path (str|None): a path of a json file which is used to save test output infomation fo model.

resume (bool): start training from last checkpoint.

validset_divide (bool): whether to split validset. if True, the dataset is split to trainset-validset-testset. if False, the dataset is split to trainset-testset.

test_step (int): the epoch number of training after which conducts the evaluation on test.

best_folds_accuracy (list|None): when running k-fold cross validation, this keeps the accuracy of folds that already run.

class mwptoolkit.trainer.supervised_trainer.**HMSTrainer**(*config*, *model*, *dataloader*, *evaluator*)

Bases: *mwptoolkit.trainer.supervised_trainer.GTSTrainer*

Parameters

- **config** (*config*) – An instance object of Config, used to record parameter information.
- **model** (*Model*) – An object of deep-learning model.
- **dataloader** (*Dataloader*) – dataloader object.
- **evaluator** (*Evaluator*) – evaluator object.

expected that config includes these parameters below:

learning_rate (float): learning rate of model.

embedding_learning_rate (float): learning rate of embedding module.

train_batch_size (int): the training batch size.

step_size (int): step_size of scheduler.

epoch_nums (int): number of epochs.

trained_model_path (str): a path of file which is used to save parameters of best model.

checkpoint_path (str): a path of file which is used save checkpoint of training progress.

output_path (str|None): a path of a json file which is used to save test output infomation fo model.

resume (bool): start training from last checkpoint.

validset_divide (bool): whether to split validset. if True, the dataset is split to trainset-validset-testset. if False, the dataset is split to trainset-testset.

test_step (int): the epoch number of training after which conducts the evaluation on test.

best_folds_accuracy (list|None): when running k-fold cross validation, this keeps the accuracy of folds that already run.

class mwptoolkit.trainer.supervised_trainer.**MWPBertTrainer**(*config*, *model*, *dataloader*, *evaluator*)

Bases: *mwptoolkit.trainer.supervised_trainer.GTSTrainer*

Parameters

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

- **model** (*Model*) – An object of deep-learning model.
- **dataloader** (*Dataloader*) – dataloader object.
- **evaluator** (*Evaluator*) – evaluator object.

expected that config includes these parameters below:

learning_rate (float): learning rate of model.

embedding_learning_rate (float): learning rate of embedding module.

train_batch_size (int): the training batch size.

step_size (int): step_size of scheduler.

epoch_nums (int): number of epochs.

trained_model_path (str): a path of file which is used to save parameters of best model.

checkpoint_path (str): a path of file which is used save checkpoint of training progress.

output_path (str|None): a path of a json file which is used to save test output infomation fo model.

resume (bool): start training from last checkpoint.

validset_divide (bool): whether to split validset. if True, the dataset is split to trainset-validset-testset. if False, the dataset is split to trainset-testset.

test_step (int): the epoch number of training after which conducts the evaluation on test.

best_folds_accuracy (list|None): when running k-fold cross validation, this keeps the accuracy of folds that already run.

class mwptoolkit.trainer.supervised_trainer.**MultiEncDecTrainer**(*config*, *model*, *dataloader*, *evaluator*)

Bases: *mwptoolkit.trainer.supervised_trainer.GTSTrainer*

multiencdec trainer, used to implement training, testing, parameter searching for deep-learning model MultiE&D.

example of instantiation:

```
>>> trainer = MultiEncDecTrainer(config, model, dataloader, evaluator)
```

for training:

```
>>> trainer.fit()
```

for testing:

```
>>> trainer.test()
```

for parameter searching:

```
>>> trainer.param_search()
```

Parameters

- **config** (*config*) – An instance object of Config, used to record parameter information.
- **model** (*Model*) – An object of deep-learning model.
- **dataloader** (*Dataloader*) – dataloader object.
- **evaluator** (*Evaluator*) – evaluator object.

expected that config includes these parameters below:

learning_rate (float): learning rate of model.

train_batch_size (int): the training batch size.

step_size (int): step_size of scheduler.

epoch_nums (int): number of epochs.

trained_model_path (str): a path of file which is used to save parameters of best model.

checkpoint_path (str): a path of file which is used save checkpoint of training progress.

output_path (str|None): a path of a json file which is used to save test output infomation fo model.

resume (bool): start training from last checkpoint.

validset_divide (bool): whether to split validset. if True, the dataset is split to trainset-validset-testset. if False, the dataset is split to trainset-testset.

test_step (int): the epoch number of training after which conducts the evaluation on test.

best_folds_accuracy (list|None): when running k-fold cross validation, this keeps the accuracy of folds that already run.

class mwptoolkit.trainer.supervised_trainer.**PretrainSeq2SeqTrainer**(*config*, *model*, *dataloader*, *evaluator*)

Bases: *mwptoolkit.trainer.supervised_trainer.SupervisedTrainer*

Parameters

- **config** (*config*) – An instance object of Config, used to record parameter information.
- **model** (*Model*) – An object of deep-learning model.
- **dataloader** (*Dataloader*) – dataloader object.
- **evaluator** (*Evaluator*) – evaluator object.

expected that config includes these parameters below:

learning_rate (float): learning rate of model

train_batch_size (int): the training batch size.

epoch_nums (int): number of epochs.

trained_model_path (str): a path of file which is used to save parameters of best model.

checkpoint_path (str): a path of file which is used save checkpoint of training progress.

output_path (str|None): a path of a json file which is used to save test output infomation fo model.

resume (bool): start training from last checkpoint.

validset_divide (bool): whether to split validset. if True, the dataset is split to trainset-validset-testset. if False, the dataset is split to trainset-testset.

test_step (int): the epoch number of training after which conducts the evaluation on test.

best_folds_accuracy (list|None): when running k-fold cross validation, this keeps the accuracy of folds that already run.

class mwptoolkit.trainer.supervised_trainer.**PretrainTRNNTrainer**(*config*, *model*, *dataloader*, *evaluator*)

Bases: *mwptoolkit.trainer.supervised_trainer.TRNNTrainer*

Parameters

- **config** (*config*) – An instance object of Config, used to record parameter information.
- **model** (*Model*) – An object of deep-learning model.
- **dataloader** (*Dataloader*) – dataloader object.
- **evaluator** (*Evaluator*) – evaluator object.

expected that config includes these parameters below:

seq2seq_learning_rate (float): learning rate of seq2seq module.

ans_learning_rate (float): learning rate of answer module.

train_batch_size (int): the training batch size.

step_size (int): step_size of scheduler.

epoch_nums (int): number of epochs.

trained_model_path (str): a path of file which is used to save parameters of best model.

checkpoint_path (str): a path of file which is used save checkpoint of training progress.

output_path (str|None): a path of a json file which is used to save test output infomation fo model.

resume (bool): start training from last checkpoint.

validset_divide (bool): whether to split validset. if True, the dataset is split to trainset-validset-testset. if False, the dataset is split to trainset-testset.

test_step (int): the epoch number of training after which conducts the evaluation on test.

best_folds_accuracy (list|None): when running k-fold cross validation, this keeps the accuracy of folds that already run.

class mwptoolkit.trainer.supervised_trainer.**SAUSolverTrainer**(*config*, *model*, *dataloader*, *evaluator*)

Bases: *mwptoolkit.trainer.supervised_trainer.GTSTrainer*

sausolver trainer, used to implement training, testing, parameter searching for deep-learning model SAUSolver.

example of instantiation:

```
>>> trainer = SAUSolverTrainer(config, model, dataloader, evaluator)
```

for training:

```
>>> trainer.fit()
```

for testing:

```
>>> trainer.test()
```

for parameter searching:

```
>>> trainer.param_search()
```

Parameters

- **config** (*config*) – An instance object of Config, used to record parameter information.
- **model** (*Model*) – An object of deep-learning model.

- **dataloader** (*Dataloader*) – dataloader object.
- **evaluator** (*Evaluator*) – evaluator object.

expected that config includes these parameters below:

learning_rate (float): learning rate of model.

train_batch_size (int): the training batch size.

step_size (int): step_size of scheduler.

epoch_nums (int): number of epochs.

trained_model_path (str): a path of file which is used to save parameters of best model.

checkpoint_path (str): a path of file which is used save checkpoint of training progress.

output_path (str|None): a path of a json file which is used to save test output infomation fo model.

resume (bool): start training from last checkpoint.

validset_divide (bool): whether to split validset. if True, the dataset is split to trainset-validset-testset. if False, the dataset is split to trainset-testset.

test_step (int): the epoch number of training after which conducts the evaluation on test.

best_folds_accuracy (list|None): when running k-fold cross validation, this keeps the accuracy of folds that already run.

class mwptoolkit.trainer.supervised_trainer.**SalignedTrainer**(*config*, *model*, *dataloader*, *evaluator*)

Bases: *mwptoolkit.trainer.supervised_trainer.SupervisedTrainer*

saligned trainer, used to implement training, testing, parameter searching for deep-learning model S-aligned.

example of instantiation:

```
>>> trainer = SalignedTrainer(config, model, dataloader, evaluator)
```

for training:

```
>>> trainer.fit()
```

for testing:

```
>>> trainer.test()
```

for parameter searching:

```
>>> trainer.param_search()
```

Parameters

- **config** (*config*) – An instance object of Config, used to record parameter information.
- **model** (*Model*) – An object of deep-learning model.
- **dataloader** (*Dataloader*) – dataloader object.
- **evaluator** (*Evaluator*) – evaluator object.

expected that config includes these parameters below:

learning_rate (float): learning rate of model

train_batch_size (int): the training batch size.

epoch_nums (int): number of epochs.

step_size (int): step_size of scheduler.

trained_model_path (str): a path of file which is used to save parameters of best model.

checkpoint_path (str): a path of file which is used save checkpoint of training progress.

output_path (str|None): a path of a json file which is used to save test output infomation fo model.

resume (bool): start training from last checkpoint.

validset_divide (bool): whether to split validset. if True, the dataset is split to trainset-validset-testset. if False, the dataset is split to trainset-testset.

test_step (int): the epoch number of training after which conducts the evaluation on test.

best_folds_accuracy (list|None): when running k-fold cross validation, this keeps the accuracy of folds that already run.

adjust_equ(*op_target*, *eq_len*, *num_list*)

evaluate(*eval_set*)

evaluate model.

Parameters **eval_set** (*str*) – [valid | test], the dataset for evaluation.

Returns equation accuracy, value accuracy, count of evaluated datas, formatted time string of evaluation time.

Return type tuple(float,float,int,str)

fit()

train model.

test()

test model.

class mwptoolkit.trainer.supervised_trainer.**SupervisedTrainer**(*config*, *model*, *dataloader*, *evaluator*)

Bases: *mwptoolkit.trainer.abstract_trainer.AbstractTrainer*

supervised trainer, used to implement training, testing, parameter searching in supervised learning.

example of instantiation:

```
>>> trainer = SupervisedTrainer(config, model, dataloader, evaluator)
```

for training:

```
>>> trainer.fit()
```

for testing:

```
>>> trainer.test()
```

for parameter searching:

```
>>> trainer.param_search()
```

Parameters

- **config** (*config*) – An instance object of Config, used to record parameter information.
- **model** (*Model*) – An object of deep-learning model.
- **dataloader** (*Dataloader*) – dataloader object.
- **evaluator** (*Evaluator*) – evaluator object.

expected that config includes these parameters below:

learning_rate (float): learning rate of model

train_batch_size (int): the training batch size.

epoch_nums (int): number of epochs.

trained_model_path (str): a path of file which is used to save parameters of best model.

checkpoint_path (str): a path of file which is used save checkpoint of training progress.

output_path (str|None): a path of a json file which is used to save test output infomation fo model.

resume (bool): start training from last checkpoint.

validset_divide (bool): whether to split validset. if True, the dataset is split to trainset-validset-testset. if False, the dataset is split to trainset-testset.

test_step (int): the epoch number of training after which conducts the evaluation on test.

best_folds_accuracy (list|None): when running k-fold cross validation, this keeps the accuracy of folds that already run.

evaluate(*eval_set*)

evaluate model.

Parameters **eval_set** (*str*) – [valid | test], the dataset for evaluation.

Returns equation accuracy, value accuracy, count of evaluated datas, formatted time string of evaluation time.

Return type tuple(float,float,int,str)

fit()

train model.

param_search()

hyper-parameter search.

test()

test model.

class mwptoolkit.trainer.supervised_trainer.**TRNNTrainer**(*config*, *model*, *dataloader*, *evaluator*)

Bases: *mwptoolkit.trainer.supervised_trainer.SupervisedTrainer*

trnn trainer, used to implement training, testing, parameter searching for deep-learning model TRNN.

example of instantiation:

```
>>> trainer = TRNNTrainer(config, model, dataloader, evaluator)
```

for training:

```
>>> trainer.fit()
```

for testing:

```
>>> trainer.test()
```

for parameter searching:

```
>>> trainer.param_search()
```

Parameters

- **config** (*config*) – An instance object of Config, used to record parameter information.
- **model** (*Model*) – An object of deep-learning model.
- **dataloader** (*Dataloader*) – dataloader object.
- **evaluator** (*Evaluator*) – evaluator object.

expected that config includes these parameters below:

seq2seq_learning_rate (float): learning rate of seq2seq module.

ans_learning_rate (float): learning rate of answer module.

train_batch_size (int): the training batch size.

step_size (int): step_size of scheduler.

epoch_nums (int): number of epochs.

trained_model_path (str): a path of file which is used to save parameters of best model.

checkpoint_path (str): a path of file which is used save checkpoint of training progress.

output_path (str|None): a path of a json file which is used to save test output infomation fo model.

resume (bool): start training from last checkpoint.

validset_divide (bool): whether to split validset. if True, the dataset is split to trainset-validset-testset. if False, the dataset is split to trainset-testset.

test_step (int): the epoch number of training after which conducts the evaluation on test.

best_folds_accuracy (list|None): when running k-fold cross validation, this keeps the accuracy of folds that already run.

evaluate(*eval_set*)

evaluate model.

Parameters **eval_set** (*str*) – [valid | test], the dataset for evaluation.

Returns equation accuracy, value accuracy, seq2seq module accuracy, answer module accuracy, count of evaluated datas, formatted time string of evaluation time.

Return type tuple(float,float,float,float,int,str)

fit()

train model.

param_search()

hyper-parameter search.

test()

test model.

class mwptoolkit.trainer.supervised_trainer.**TSNTrainer**(*config*, *model*, *dataloader*, *evaluator*)

Bases: *mwptoolkit.trainer.abstract_trainer.AbstractTrainer*

tsn trainer, used to implement training, testing, parameter searching for deep-learning model TSN.

example of instantiation:

```
>>> trainer = TSNTrainer(config, model, dataloader, evaluator)
```

for training:

```
>>> trainer.fit()
```

for testing:

```
>>> trainer.test()
```

for parameter searching:

```
>>> trainer.param_search()
```

Parameters

- **config** (*config*) – An instance object of Config, used to record parameter information.
- **model** (*Model*) – An object of deep-learning model.
- **dataloader** (*Dataloader*) – dataloader object.
- **evaluator** (*Evaluator*) – evaluator object.

expected that config includes these parameters below:

learning_rate (float): learning rate of model

train_batch_size (int): the training batch size.

epoch_nums (int): number of epochs.

step_size (int): step_size of scheduler.

trained_model_path (str): a path of file which is used to save parameters of best model.

checkpoint_path (str): a path of file which is used save checkpoint of training progress.

output_path (str|None): a path of a json file which is used to save test output infomation fo model.

resume (bool): start training from last checkpoint.

validset_divide (bool): whether to split validset. if True, the dataset is split to trainset-validset-testset. if False, the dataset is split to trainset-testset.

test_step (int): the epoch number of training after which conducts the evaluation on test.

best_folds_accuracy (list|None): when running k-fold cross validation, this keeps the accuracy of folds that already run.

evaluate_student(*eval_set*)

evaluate student net.

Parameters **eval_set** (*str*) – [valid | test], the dataset for evaluation.

Returns equation accuracy, value accuracy, equation accuracy of student net 1, value accuracy of student net 1, equation accuracy of student net 2, value accuracy of student net 2, count of evaluated datas, formatted time string of evaluation time.

Return type tuple(float,float,float,float,float,float,int,str)

evaluate_teacher(*eval_set*)

evaluate teacher net.

Parameters **eval_set** (*str*) – [valid | test], the dataset for evaluation.

Returns equation accuracy, value accuracy, count of evaluated datas, formatted time string of evaluation time.

Return type tuple(float,float,int,str)

fit()

train model.

test()

test model.

class mwptoolkit.trainer.supervised_trainer.**TreeLSTMTrainer**(*config*, *model*, *dataloader*, *evaluator*)

Bases: *mwptoolkit.trainer.abstract_trainer.AbstractTrainer*

treelstm trainer, used to implement training, testing, parameter searching for deep-learning model TreeLSTM.

example of instantiation:

```
>>> trainer = TreeLSTMTrainer(config, model, dataloader, evaluator)
```

for training:

```
>>> trainer.fit()
```

for testing:

```
>>> trainer.test()
```

for parameter searching:

```
>>> trainer.param_search()
```

Parameters

- **config** (*config*) – An instance object of Config, used to record parameter information.
- **model** (*Model*) – An object of deep-learning model.
- **dataloader** (*Dataloader*) – dataloader object.
- **evaluator** (*Evaluator*) – evaluator object.

expected that config includes these parameters below:

learning_rate (float): learning rate of model.

train_batch_size (int): the training batch size.

step_size (int): step_size of scheduler.

epoch_nums (int): number of epochs.

trained_model_path (str): a path of file which is used to save parameters of best model.

checkpoint_path (str): a path of file which is used save checkpoint of training progress.

output_path (str|None): a path of a json file which is used to save test output infomation fo model.

resume (bool): start training from last checkpoint.

validset_divide (bool): whether to split validset. if True, the dataset is split to trainset-validset-testset. if False, the dataset is split to trainset-testset.

test_step (int): the epoch number of training after which conducts the evaluation on test.

best_folds_accuracy (list|None): when running k-fold cross validation, this keeps the accuracy of folds that already run.

evaluate(*eval_set*)

evaluate model.

Parameters **eval_set** (*str*) – [valid | test], the dataset for evaluation.

Returns equation accuracy, value accuracy, count of evaluated datas, formatted time string of evaluation time.

Return type tuple(float,float,int,str)

fit()

train model.

param_search()

test()

test model.

7.3 mwptoolkit.trainer.template_trainer

class mwptoolkit.trainer.template_trainer.**TemplateTrainer**(*config*, *model*, *dataloader*, *evaluator*)

Bases: *mwptoolkit.trainer.abstract_trainer.AbstractTrainer*

template trainer.

you need implement:

TemplateTrainer._build_optimizer()

TemplateTrainer._save_checkpoint()

TemplateTrainer._load_checkpoint()

TemplateTrainer._train_batch()

TemplateTrainer._eval_batch()

Parameters

- **config** (*config*) – An instance object of Config, used to record parameter information.
- **model** (*Model*) – An object of deep-learning model.

- **dataloader** (*Dataloader*) – dataloader object.
- **evaluator** (*Evaluator*) – evaluator object.

expected that config includes these parameters below:

test_step (int): the epoch number of training after which conducts the evaluation on test.

best_folds_accuracy (list|None): when running k-fold cross validation, this keeps the accuracy of folds that already run.

evaluate(*eval_set*)

fit()

test()

CHAPTER

EIGHT

MWPTOOLKIT.UTILS

8.1 mwptoolkit.utils.data_structure

class mwptoolkit.utils.data_structure.**AbstractTree**

Bases: object

equ2tree()

tree2equ()

class mwptoolkit.utils.data_structure.**BinaryTree**(*root_node=None*)

Bases: *mwptoolkit.utils.data_structure.AbstractTree*

binary tree

equ2tree(*equ_list*, *out_idx2symbol*, *op_list*, *input_var*, *emb*)

equ2tree_(*equ_list*)

tree2equ(*node*)

class mwptoolkit.utils.data_structure.**DependencyNode**(*node_value*, *position*, *relation*, *is_leaf=True*)

Bases: object

add_left_node(*node*)

add_right_node(*node*)

class mwptoolkit.utils.data_structure.**DependencyTree**(*root_node=None*)

Bases: object

sentence2tree(*sentence*, *dependency_info*)

dependency info [relation,child,father]

class mwptoolkit.utils.data_structure.**GoldTree**(*root_node=None*, *gold_ans=None*)

Bases: *mwptoolkit.utils.data_structure.AbstractTree*

equ2tree(*equ_list*, *out_idx2symbol*, *op_list*, *num_list*, *ans*)

is_equal(*v1*, *v2*)

is_float(*num_str*, *num_list*)

is_in_rel_quants(*value*, *rel_quants*)

lca(*root*, *va*, *vb*, *parent*)

query(*va*, *vb*)

class mwptoolkit.utils.data_structure.**Node**(*node_value*, *isleaf=True*)

Bases: object

node

set_left_node(*node*)

set_right_node(*node*)

class mwptoolkit.utils.data_structure.**PrefixTree**(*root_node*)

Bases: *mwptoolkit.utils.data_structure.BinaryTree*

prefix2tree(*equ_list*)

class mwptoolkit.utils.data_structure.**Tree**

Bases: object

add_child(*c*)

to_list(*out_idx2symbol*)

to_string()

8.2 mwptoolkit.utils.enum_type

class mwptoolkit.utils.enum_type.**DatasetLanguage**

Bases: object

dataset language

en = 'en'

zh = 'zh'

class mwptoolkit.utils.enum_type.**DatasetName**

Bases: object

dataset name

SVAMP = 'SVAMP'

alg514 = 'alg514'

ape200k = 'ape200k'

asdiv_a = 'asdiv-a'

draw = 'draw'

hmwp = 'hmwp'

math23k = 'math23k'

mawps = 'mawps'

mawps_asdiv_a_svamp = 'mawps_asdiv-a_svamp'

mawps_single = 'mawps-single'

class mwptoolkit.utils.enum_type.DatasetType

Bases: object

dataset type

Test = 'test'

Train = 'train'

Valid = 'valid'

class mwptoolkit.utils.enum_type.EPT

Bases: object

ARG_CON = 'CONST:'

ARG_CON_ID = 0

ARG_MEM = 'MEMORY:'

ARG_MEM_ID = 2

ARG_NUM = 'NUMBER:'

ARG_NUM_ID = 1

ARG_TOKENS = ['CONST:', 'NUMBER:', 'MEMORY:']

ARG_UNK = 'UNK'

ARG_UNK_ID = 0

ARITY_MAP = {(2, False): ['+', '-', '*', '/', '^'], (2, True): ['=']}

CON_PREFIX = 'C_'

FIELD_EXPR_GEN = 'expr_gen'

FIELD_EXPR_PTR = 'expr_ptr'

FIELD_OP_GEN = 'op_gen'

FOLLOWING_ZERO_PATTERN = re.compile('(\\d+|\\d+_[0-9]*[1-9])_?(0+|0{4}\\d+)$')

FORMAT_MEM = 'M_%02d'

FORMAT_NUM = 'N_%02d'

FORMAT_VAR = 'X_%01d'

FRACTIONAL_PATTERN = re.compile('(\\d+/\\d+)')

FUN_END_EQN = '__DONE'

FUN_END_EQN_ID = 1

FUN_EQ_SGN_ID = 3

FUN_NEW_EQN = '__NEW_EQN'

FUN_NEW_EQN_ID = 0

FUN_NEW_VAR = '__NEW_VAR'

FUN_NEW_VAR_ID = 2

FUN_TOKENS = ['__NEW_EQN', '__DONE', '__NEW_VAR']

FUN_TOKENS_WITH_EQ = ['__NEW_EQN', '__DONE', '__NEW_VAR', '=']

IN_EQN = 'equation'

IN_TNPAD = 'text_numpad'

IN_TNUM = 'text_num'

IN_TPAD = 'text_pad'

IN_TXT = 'text'

MEM_MAX = 32

MEM_PREFIX = 'M_'

MODEL_EXPR_PTR_TRANS = 'ept'

MODEL_EXPR_TRANS = 'expr'

MODEL_VANILLA_TRANS = 'vanilla'

MULTIPLES = ['once', 'twice', 'thrice', 'double', 'triple', 'quadruple', 'dozen',
'half', 'quarter', 'doubled', 'tripled', 'quadrupled', 'halved', 'quartered']

NEG_INF = -inf

NUMBER_AND_FRACTION_PATTERN =
re.compile('((\\d+/\\d+)|([+\\-]?(\\d{1,3}(,\\d{3})+|\\d+)(\\.\\d+)?))')

NUMBER_PATTERN = re.compile('([+\\-]?(\\d{1,3}(,\\d{3})+|\\d+)(\\.\\d+)?)')

NUMBER_READINGS = {'billion': 1000000000, 'billionth': 1000000000, 'double': 2,
'doubled': 2, 'dozen': 12, 'eight': 8, 'eighteen': 18, 'eighteenth': 18,
'eighth': 8, 'eightieth': 80, 'eighty': 80, 'eleven': 11, 'eleventh': 11,
'fifteen': 15, 'fifteenth': 15, 'fifth': 5, 'fiftieth': 50, 'fifty': 50,
'five': 5, 'forth': 4, 'fortieth': 40, 'forty': 40, 'four': 4, 'fourteen': 14,
'fourteenth': 14, 'fourth': 4, 'half': 0.5, 'halved': 0.5, 'hundred': 100,
'hundredth': 100, 'million': 1000000, 'millionth': 1000000, 'nine': 9,
'nineteen': 19, 'nineteenth': 19, 'ninetieth': 90, 'ninety': 90, 'ninth': 9,
'once': 1, 'one': 1, 'quadruple': 4, 'quadrupled': 4, 'quarter': 0.25,
'quartered': 0.25, 'seven': 7, 'seventeen': 17, 'seventeenth': 17, 'seventh':
7, 'seventieth': 70, 'seventy': 70, 'six': 6, 'sixteen': 16, 'sixteenth': 16,
'sixth': 6, 'sixtieth': 60, 'sixty': 60, 'ten': 10, 'tenth': 10, 'third': 3,
'thirteen': 13, 'thirteenth': 13, 'thirtieth': 30, 'thirty': 30, 'thousand':
1000, 'thousandth': 1000, 'three': 3, 'thrice': 3, 'triple': 3, 'tripled': 3,
'twelfth': 12, 'twelve': 12, 'twentieth': 20, 'twenty': 20, 'twice': 2, 'two':
2, 'zero': 0}

NUM_MAX = 32

NUM_PREFIX = 'N_'

NUM_TOKEN = '[N]'

OPERATORS = {'*': {'arity': 2, 'commutable': True, 'convert': <function EPT.<lambda>>, 'top_level': False}, '+': {'arity': 2, 'commutable': True, 'convert': <function EPT.<lambda>>, 'top_level': False}, '-': {'arity': 2, 'commutable': False, 'convert': <function EPT.<lambda>>, 'top_level': False}, '/': {'arity': 2, 'commutable': False, 'convert': <function EPT.<lambda>>, 'top_level': False}, '=': {'arity': 2, 'commutable': True, 'convert': <function EPT.<lambda>>, 'top_level': True}, '^': {'arity': 2, 'commutable': False, 'convert': <function EPT.<lambda>>, 'top_level': False}}

OPERATOR_PRECEDENCE = {'*': 3, '+': 2, '-': 2, '/': 3, '=': 1, '^': 4}

PAD_ID = -1

PLURAL_FORMS = [('ies', 'y'), ('ves', 'f'), ('s', '')]

POS_INF = inf

PREP_KEY_ANS = 1

PREP_KEY_EQN = 0

PREP_KEY_MEM = 2

SEQ_END_EQN = '__DONE'

SEQ_END_EQN_ID = 1

SEQ_EQ_SGN_ID = 3

SEQ_GEN_NUM_ID = 4

SEQ_GEN_VAR_ID = 36

SEQ_NEW_EQN = '__NEW_EQN'

SEQ_NEW_EQN_ID = 0

SEQ_PTR_NUM = '__NUM'

SEQ_PTR_NUM_ID = 4

SEQ_PTR_TOKENS = ['__NEW_EQN', '__DONE', 'UNK', '=', '__NUM', '__VAR']

SEQ_PTR_VAR = '__VAR'

SEQ_PTR_VAR_ID = 5

SEQ_TOKENS = ['__NEW_EQN', '__DONE', 'UNK', '=']

SEQ_UNK_TOK = 'UNK'

SEQ_UNK_TOK_ID = 2

SPIECE_UNDERLINE = ''

TOP_LEVEL_CLASSES = ['Eq']

VAR_MAX = 2

VAR_PREFIX = 'X_'

class mwptoolkit.utils.enum_type.**FixType**

Bases: `object`

equation fix type

Infix = 'infix'

MultiWayTree = 'multi_way_tree'

Nonfix = None

Postfix = 'postfix'

Prefix = 'prefix'

class mwptoolkit.utils.enum_type.**MaskSymbol**

Bases: `object`

number mask type

NUM = 'NUM'

alphabet = 'alphabet'

number = 'number'

class mwptoolkit.utils.enum_type.**NumMask**

Bases: `object`

number mask symbol list

NUM = ['NUM', 'NUM']

alphabet = ['NUM_a', 'NUM_b', 'NUM_c', 'NUM_d', 'NUM_e', 'NUM_f', 'NUM_g', 'NUM_h', 'NUM_i', 'NUM_j', 'NUM_k', 'NUM_l', 'NUM_m', 'NUM_n', 'NUM_o', 'NUM_p', 'NUM_q', 'NUM_r', 'NUM_s', 'NUM_t', 'NUM_u', 'NUM_v', 'NUM_w', 'NUM_x', 'NUM_y', 'NUM_z']

number = ['NUM_0', 'NUM_1', 'NUM_2', 'NUM_3', 'NUM_4', 'NUM_5', 'NUM_6', 'NUM_7', 'NUM_8', 'NUM_9', 'NUM_10', 'NUM_11', 'NUM_12', 'NUM_13', 'NUM_14', 'NUM_15', 'NUM_16', 'NUM_17', 'NUM_18', 'NUM_19', 'NUM_20', 'NUM_21', 'NUM_22', 'NUM_23', 'NUM_24', 'NUM_25', 'NUM_26', 'NUM_27', 'NUM_28', 'NUM_29', 'NUM_30', 'NUM_31', 'NUM_32', 'NUM_33', 'NUM_34', 'NUM_35', 'NUM_36', 'NUM_37', 'NUM_38', 'NUM_39', 'NUM_40', 'NUM_41', 'NUM_42', 'NUM_43', 'NUM_44', 'NUM_45', 'NUM_46', 'NUM_47', 'NUM_48', 'NUM_49', 'NUM_50', 'NUM_51', 'NUM_52', 'NUM_53', 'NUM_54', 'NUM_55', 'NUM_56', 'NUM_57', 'NUM_58', 'NUM_59', 'NUM_60', 'NUM_61', 'NUM_62', 'NUM_63', 'NUM_64', 'NUM_65', 'NUM_66', 'NUM_67', 'NUM_68', 'NUM_69', 'NUM_70', 'NUM_71', 'NUM_72', 'NUM_73', 'NUM_74', 'NUM_75', 'NUM_76', 'NUM_77', 'NUM_78', 'NUM_79', 'NUM_80', 'NUM_81', 'NUM_82', 'NUM_83', 'NUM_84', 'NUM_85', 'NUM_86', 'NUM_87', 'NUM_88', 'NUM_89', 'NUM_90', 'NUM_91', 'NUM_92', 'NUM_93', 'NUM_94', 'NUM_95', 'NUM_96', 'NUM_97', 'NUM_98', 'NUM_99']

class mwptoolkit.utils.enum_type.Operators

Bases: object

operators in equation.

Multi = ['+', '-', '*', '/', '^', '=', '<BRG>']

Single = ['+', '-', '*', '/', '^']

class mwptoolkit.utils.enum_type.SpecialTokens

Bases: object

special tokens

BRG_TOKEN = '<BRG>'

EOS_TOKEN = '<EOS>'

NON_TOKEN = '<NON>'

OPT_TOKEN = '<OPT>'

PAD_TOKEN = '<PAD>'

SOS_TOKEN = '<SOS>'

UNK_TOKEN = '<UNK>'

class mwptoolkit.utils.enum_type.SupervisingMode

Bases: object

supervising mode

fully_supervised = 'fully_supervised'

weakly_supervised = ['fix', 'mafix', 'reinforce', 'mapo']

class mwptoolkit.utils.enum_type.TaskType

Bases: object

task type

MultiEquation = 'multi_equation'

SingleEquation = 'single_equation'

8.3 mwptoolkit.utils.logger

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

8.4 mwptoolkit.utils.preprocess_tool

8.4.1 mwptoolkit.utils.preprocess_tool.dataset_operator

mwptoolkit.utils.preprocess_tool.dataset_operator.**ept_preprocess**(*datas*, *dataset_name*)

mwptoolkit.utils.preprocess_tool.dataset_operator.**id_reedit**(*trainset*, *validset*, *testset*)

if some datas of a dataset hava the same id, re-edit the id for differentiate them.

example: There are two datas have the same id 709356. Make one of them be 709356 and the other be 709356-1.

mwptoolkit.utils.preprocess_tool.dataset_operator.**preprocess_ept_dataset_**(*train_datas*, *valid_datas*, *test_datas*, *dataset_name*)

mwptoolkit.utils.preprocess_tool.dataset_operator.**refine_formula_as_prefix**(*item*, *numbers*, *dataset_name*)

8.4.2 mwptoolkit.utils.preprocess_tool.equation_operator

mwptoolkit.utils.preprocess_tool.equation_operator.**EN_rule1_stat**(*datas*, *sample_k=100*)

equation norm rule1

Parameters

- **datas** (*list*) – dataset.
- **sample_k** (*int*) – number of random sample.

Returns classified equations. equivalent equations will be in the same class.

Return type (list)

mwptoolkit.utils.preprocess_tool.equation_operator.**EN_rule2**(*equ_list*)

equation norm rule2

Parameters **equ_list** (*list*) – equation.

Returns equivalent equation.

Return type list

mwptoolkit.utils.preprocess_tool.equation_operator.**from_infix_to_multi_way_tree**(*expression*)

mwptoolkit.utils.preprocess_tool.equation_operator.**from_infix_to_postfix**(*expression*)
convert infix equation to postfix equation.

Parameters **expression** (*list*) – infix expression.

Returns postfix expression.

Return type (list)

mwptoolkit.utils.preprocess_tool.equation_operator.**from_infix_to_prefix**(*expression*)
convert infix equation to prefix equation

Parameters **expression** (*list*) – infix expression.

Returns prefix expression.

Return type (list)

mwptoolkit.utils.preprocess_tool.equation_operator.**from_postfix_to_infix**(*expression*)
convert postfix equation to infix equation

Parameters **expression** (*list*) – postfix expression.

Returns infix expression.

Return type (list)

mwptoolkit.utils.preprocess_tool.equation_operator.**from_postfix_to_prefix**(*expression*)
convert postfix equation to prefix equation

Parameters **expression** (*list*) – postfix expression.

Returns prefix expression.

Return type (list)

mwptoolkit.utils.preprocess_tool.equation_operator.**from_prefix_to_infix**(*expression*)
convert prefix equation to infix equation

Parameters **expression** (*list*) – prefix expression.

Returns infix expression.

Return type (list)

mwptoolkit.utils.preprocess_tool.equation_operator.**from_prefix_to_postfix**(*expression*)
convert prefix equation to postfix equation

Parameters **expression** (*list*) – prefix expression.

Returns postfix expression.

Return type (list)

mwptoolkit.utils.preprocess_tool.equation_operator.**infix_to_postfix**(*equation*, *free_symbols: list*, *join_output: bool = True*)

mwptoolkit.utils.preprocess_tool.equation_operator.**operator_mask**(*expression*)

mwptoolkit.utils.preprocess_tool.equation_operator.**orig_infix_to_postfix**(*equation: Union[str, List[str]]*, *number_token_map: dict*, *free_symbols: list*, *join_output: bool = True*)

Read infix equation string and convert it into a postfix string

Parameters

- **equation** (*Union[str,List[str]]*) – Either one of these. - A single string of infix equation. e.g. "5 + 4" - Tokenized sequence of infix equation. e.g. ["5", "+", "4"]
- **number_token_map** (*dict*) – Mapping from a number token to its anonymized representation (e.g. N_0)
- **free_symbols** (*list*) – List of free symbols (for return)
- **join_output** (*bool*) – True if the output need to be joined. Otherwise, this method will return the tokenized postfix sequence.

Return type Union[str, List[str]]

Returns Either one of these. - A single string of postfix equation. e.g. "5 4 +" - Tokenized sequence of postfix equation. e.g. ["5", "4", "+"]

mwptoolkit.utils.preprocess_tool.equation_operator.**postfix_parser**(*equation*, *memory: list*) → int

Read Op-token postfix equation and transform it into Expression-token sequence.

Parameters

- **equation** (*List[Union[str,Tuple[str,Any]]]*) – List of op-tokens to be parsed into a Expression-token sequence Item of this list should be either - an operator string - a tuple of (operand source, operand value)
- **memory** (*list*) – List where previous execution results of expressions are stored

Return type int

Returns Size of stack after processing. Value 1 means parsing was done without any free expression.

mwptoolkit.utils.preprocess_tool.equation_operator.**trans_symbol_2_number**(*equ_list*, *num_list*)

transfer mask symbol in equation to number.

Parameters

- **equ_list** (*list*) – equation.
- **num_list** (*list*) – number list.

Returns equation.

Return type (list)

8.4.3 mwptoolkit.utils.preprocess_tool.number_operator

mwptoolkit.utils.preprocess_tool.number_operator.**constant_number**(*const*)

Converts number to constant symbol string (e.g. 'C_3'). To avoid sympy's automatic simplification of operation over constants.

Parameters **const** (*Union[str,int,float,Expr]*) – constant value to be converted.

Returns (str) Constant symbol string represents given constant.

mwptoolkit.utils.preprocess_tool.number_operator.**english_word_2_num**(*sentence_list*, *fraction_acc=None*)

transfer english word to number.

Parameters

- **sentence_list** (*list*) – list of words.
- **fraction_acc** (*int|None*) – the accuracy to transfer fraction to float, if None, not to match fraction expression.

Returns transfered sentence.

Return type (list)

mwptoolkit.utils.preprocess_tool.number_operator.**fraction_word_to_num**(*number_sentence*)

transfer english expression of fraction to number. numerator and denominator are not more than 10.

Parameters **number_sentence** (*str*) – english expression.

Returns number

Return type (float)

mwptoolkit.utils.preprocess_tool.number_operator.**joint_number**(*text_list*)

joint fraction number

Parameters **text_list** (*list*) – text list.

Returns processed text list.

Return type (list)

mwptoolkit.utils.preprocess_tool.number_operator.**joint_number_**(*text_list*)

mwptoolkit.utils.preprocess_tool.number_operator.**split_number**(*text_list*)

separate number expression from other characters.

Parameters **text_list** (*list*) – text list.

Returns processed text list.

Return type (list)

mwptoolkit.utils.preprocess_tool.number_operator.**trans_symbol_2_number**(*equ_list*, *num_list*)

transfer mask symbol in equation to number.

Parameters

- **equ_list** (*list*) – equation.
- **num_list** (*list*) – number list.

Returns equation.

Return type (list)

8.4.4 mwptoolkit.utils.preprocess_tool.number_transfer

mwptoolkit.utils.preprocess_tool.number_transfer.**get_num_pos**(*input_seq*, *mask_type*, *pattern*)

mwptoolkit.utils.preprocess_tool.number_transfer.**num_transfer_alg514**(*data*, *mask_type*, *equ_split_symbol=';'*, *vocab_level='word'*)

mwptoolkit.utils.preprocess_tool.number_transfer.**num_transfer_draw**(*data*, *mask_type*, *equ_split_symbol=';'*, *vocab_level='word'*)

mwptoolkit.utils.preprocess_tool.number_transfer.**num_transfer_hmwp**(*data*, *mask_type*, *equ_split_symbol=';'*, *vocab_level='word'*)

mwptoolkit.utils.preprocess_tool.number_transfer.**num_transfer_multi**(*data*, *mask_type*, *equ_split_symbol=';'*, *vocab_level='word'*)

mwptoolkit.utils.preprocess_tool.number_transfer.**number_transfer**(*datas*, *dataset_name*, *task_type*, *mask_type*, *min_generate_keep*, *linear_dataset*, *equ_split_symbol=';'*, *vocab_level='word'*)

number transfer

Parameters

- **datas** (*list*) – dataset.
- **dataset_name** (*str*) – dataset name.
- **task_type** (*str*) – [single_equation | multi_equation], task type.
- **min_generate_keep** (*int*) – generate number that count greater than the value, will be kept in output symbols.
- **equ_split_symbol** (*str*) – equation split symbol, in multiple-equation dataset, symbol to split equations, this symbol will be repalced with special token SpecialTokens.BRG.

Returns processed datas, generate number list, copy number, unk symbol list.

Return type tuple(list,list,int,list)

mwptoolkit.utils.preprocess_tool.number_transfer.**number_transfer_asdiv_a**(*data*, *mask_type*, *linear*, *vocab_level='word'*)

mwptoolkit.utils.preprocess_tool.number_transfer.**number_transfer_math23k**(*data*, *mask_type*, *linear*, *vocab_level='word'*)

mwptoolkit.utils.preprocess_tool.number_transfer.**number_transfer_mawps**(*data*, *mask_type*, *linear*, *vocab_level='word'*)

mwptoolkit.utils.preprocess_tool.number_transfer.**number_transfer_mawps_single**(*data*, *mask_type*, *linear*, *vocab_level='word'*)

mwptoolkit.utils.preprocess_tool.number_transfer.**number_transfer_single**(*data*, *mask_type*, *linear*, *vocab_level='word'*)

mwptoolkit.utils.preprocess_tool.number_transfer.**number_transfer_svamp**(*data*, *mask_type*, *linear*, *vocab_level='word'*)

mwptoolkit.utils.preprocess_tool.number_transfer.**seg_and_tag_asdiv_a**(*st*, *nums_fraction*, *nums*)

mwptoolkit.utils.preprocess_tool.number_transfer.**seg_and_tag_hmwp**(*st*, *nums_fraction*, *nums*)

mwptoolkit.utils.preprocess_tool.number_transfer.**seg_and_tag_math23k**(*st*, *nums_fraction*, *nums*)

mwptoolkit.utils.preprocess_tool.number_transfer.**seg_and_tag_mawps**(*st*, *nums_fraction*, *nums*)

mwptoolkit.utils.preprocess_tool.number_transfer.**seg_and_tag_mawps_single**(*st*, *nums_fraction*, *nums*)

mwptoolkit.utils.preprocess_tool.number_transfer.**seg_and_tag_multi**(*st*, *nums_fraction*, *nums*)

mwptoolkit.utils.preprocess_tool.number_transfer.**seg_and_tag_single**(*st*, *nums_fraction*, *nums*)

mwptoolkit.utils.preprocess_tool.number_transfer.**seg_and_tag_svamp**(*st*, *nums_fraction*, *nums*)

8.4.5 mwptoolkit.utils.preprocess_tool.sentence_operator

mwptoolkit.utils.preprocess_tool.sentence_operator.**deprel_tree_to_file**(*train_datas*, *valid_datas*, *test_datas*, *path*, *language*, *use_gpu*)

save deprel tree infomation to file

mwptoolkit.utils.preprocess_tool.sentence_operator.**find_ept_numbers_in_text**(*text: str*, *append_number_token: bool = False*)

mwptoolkit.utils.preprocess_tool.sentence_operator.**get_deprel_tree**(*datas*, *language*)

mwptoolkit.utils.preprocess_tool.sentence_operator.**get_deprel_tree_**(*train_datas*, *valid_datas*, *test_datas*, *path*)

get deprel tree infomation from file

mwptoolkit.utils.preprocess_tool.sentence_operator.**get_group_nums**(*datas*, *language*, *use_gpu*)

mwptoolkit.utils.preprocess_tool.sentence_operator.**get_group_nums_**(*train_datas*, *valid_datas*, *test_datas*, *path*)

get group nums infomation from file.

mwptoolkit.utils.preprocess_tool.sentence_operator.**get_span_level_deprel_tree**(*datas*, *language*)

mwptoolkit.utils.preprocess_tool.sentence_operator.**get_span_level_deprel_tree_**(*train_datas*, *valid_datas*, *test_datas*, *path*)

mwptoolkit.utils.preprocess_tool.sentence_operator.**span_level_deprel_tree_to_file**(*train_datas*, *valid_datas*, *test_datas*, *path*, *language*, *use_gpu*)

mwptoolkit.utils.preprocess_tool.sentence_operator.**split_sentence**(*text*)
split sentence by punctuations.

8.5 mwptoolkit.utils.utils

mwptoolkit.utils.utils.**clones**(*module*, *N*)
Produce N identical layers.

mwptoolkit.utils.utils.**copy_list**(*l*)

mwptoolkit.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 Model

mwptoolkit.utils.utils.**get_trainer**(*config*)
Automatically select trainer class based on task type and model name

Parameters **config** (Config) –

Returns trainer class

Return type *SupervisedTrainer*

mwptoolkit.utils.utils.**get_trainer_**(*task_type*, *model_name*, *sup_mode*)
Automatically select trainer class based on model type and model name

Parameters

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

Returns trainer class

Return type *Trainer*

mwptoolkit.utils.utils.**get_weakly_supervised**(*supervising_mode*)

mwptoolkit.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

mwptoolkit.utils.utils.**lists2dict**(*list1*, *list2*)
convert two lists to dict, elements of first list as keys, another's as values.

mwptoolkit.utils.utils.**read_ape200k_source**(*filename*)
specially used to read data of ape200k source file

mwptoolkit.utils.utils.**read_json_data**(*filename*)
load data from a json file

mwptoolkit.utils.utils.**read_math23k_source**(*filename*)

specially used to read data of math23k source file

mwptoolkit.utils.utils.**str2float**(*v*)

convert string to float.

mwptoolkit.utils.utils.**time_since**(*s*)

compute time

Parameters **s** (*float*) – the amount of time in seconds.

Returns formatting time.

Return type (str)

mwptoolkit.utils.utils.**write_json_data**(*data*, *filename*)

write data to a json file

CHAPTER

NINE

MWPTOOLKIT.HYPER_SEARCH

mwptoolkit.hyper_search.**hyper_search_process**(*model_name*, *dataset_name*, *task_type*, *search_parameter*, *config_dict={}*)

mwptoolkit.hyper_search.**train_process**(*search_parameter*, *configs*)

CHAPTER

TEN

MWPTOOLKIT.QUICK_START

mwptoolkit.quick_start.**run_toolkit**(*model_name*, *dataset_name*, *task_type*, *config_dict={}*)

mwptoolkit.quick_start.**test_with_cross_validation**(*temp_config*)

mwptoolkit.quick_start.**test_with_train_valid_test_split**(*temp_config*)

mwptoolkit.quick_start.**train_cross_validation**(*config*)

mwptoolkit.quick_start.**train_with_cross_validation**(*temp_config*)

mwptoolkit.quick_start.**train_with_train_valid_test_split**(*temp_config*)

CHAPTER

ELEVEN

MWPTOOLKIT USAGE:

command line lookup

CHAPTER

TWELVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

m

INDEX

Symbols

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

Z