

---

# MWPToolkit

*Release 0.0.6*

”

Sep 28, 2022



## MWPTOOLKIT API:

<b>1</b>	<b>mwptoolkit.config.configuration</b>	<b>1</b>
<b>2</b>	<b>mwptoolkit.data</b>	<b>3</b>
<b>3</b>	<b>mwptoolkit.evaluate.evaluator</b>	<b>25</b>
<b>4</b>	<b>mwptoolkit.loss</b>	<b>31</b>
<b>5</b>	<b>mwptoolkit.model</b>	<b>37</b>
<b>6</b>	<b>mwptoolkit.module</b>	<b>69</b>
<b>7</b>	<b>mwptoolkit.trainer</b>	<b>123</b>
<b>8</b>	<b>mwptoolkit.utils</b>	<b>141</b>
<b>9</b>	<b>mwptoolkit.hyper_search</b>	<b>157</b>
<b>10</b>	<b>mwptoolkit.quick_start</b>	<b>159</b>
<b>11</b>	<b>MWPToolkit Usage:</b>	<b>161</b>
<b>12</b>	<b>Indices and tables</b>	<b>163</b>
	<b>Python Module Index</b>	<b>165</b>
	<b>Index</b>	<b>167</b>



## 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**()

## 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: Config, dataset: DatasetEPT*)

Bases: *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*: [Config](#), *dataset*: [DatasetHMS](#))

Bases: [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*: [Config](#), *dataset*: [DatasetMultiEncDec](#))

Bases: [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:  
                                                                                   Config,  
                                                                                   dataset:  
                                                                                   Multi-  
                                                                                   Equa-  
                                                                                   tion-  
                                                                                   Dataset)
```

Bases: [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: Config, dataset: PretrainDataset*)

Bases: [\*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: Con-  
fig, dataset: Sin-  
gleE-  
qua-  
tion-  
Dataset)
```

Bases: *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: [\*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: *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: *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\_multienccdec

**`class mwptoolkit.data.dataset.dataset_multienccdec.DatasetMultiEncDec(config)`**

Bases: *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: *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: [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: *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: [\*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_data_loader(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_data_loader_module(config: Config) → Type[Union[DataLoaderMultiEncDec, DataLoaderEPT, DataLoaderHMS, DataLoaderGPT2, PretrainDataLoader, SingleEquationDataLoader, MultiEquationDataLoader, 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: Config) → Type[Union[DatasetMultiEncDec,  
DatasetEPT, DatasetHMS, DatasetGPT2, PretrainDataset,  
SingleEquationDataset, MultiEquationDataset,  
AbstractDataset]]
```

return a dataset module according to config

**Parameters**

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

**Returns**

dataset module



## MWPTOOLKIT.EVALUATE.EVALUATOR

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

Bases: object

abstract evaluator

**result**()

**result\_multi**()

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

Bases: *AbstractEvaluator*

evaluator for infix equation sequence.

**\_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 equations.

### 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: *PostfixEvaluator*, *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 equations.

**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: *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: *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: [\*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: Thread

time-limited equation-solving mechanism 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: Config*) → Type[Union[[PrefixEvaluator](#),  
[InfixEvaluator](#), [PostfixEvaluator](#),  
[MultiWayTreeEvaluator](#), [AbstractEvaluator](#),  
[MultiEncDecEvaluator](#)]]

return a evaluator module according to config

**Parameters**

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

**Returns**

evaluator module

## 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: 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: *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: *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: [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: [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: *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: 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: Tensor, target: LongTensor) → 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`





## MWPTOOLKIT.MODEL

### 5.1 mwptoolkit.model.Seq2Seq

#### 5.1.1 mwptoolkit.model.Seq2Seq.dns

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

Bases: 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[Tensor, 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], sym-  
bol\_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 r\_t 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 r\_t1 is ‘=’, then r\_t 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: 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: Tensor, mask: 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: Tensor, fill\_value=-1*) → 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: 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[Tensor, 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: 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[Tensor, 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], sym-  
bol\_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: 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],      sym-
bol_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: 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[*Tensor, 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], sym-  
bol\_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: 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[Tensor, Tensor], 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], sym-  
bol\_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: 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[*Tensor*, *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: 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[*Tensor, 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], sym-  
bol\_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: 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[*Tensor, 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],      sym-
bol_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: 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[*Tensor*, *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], sym-  
bol\_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: 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[*Tensor, 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],      sym-  
bol_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: 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[*Tensor, 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: 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: 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[Tensor, Tensor], Tuple[Tensor, 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[Tensor, 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],      sym-  
bol_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_I, length)
```

```
mwptoolkit.model.Seq2Tree.tsn.cosine_sim(logits, logits_I)
```

```
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: 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[*Tensor, 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.multienccdec

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

Bases: 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: 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[*Tensor, 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: 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[Tensor, 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: 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[*Tensor*, *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





## 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: 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: 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: Tensor, key_value: Optional[Tensor] = None, key_ignorance_mask: Optional[Tensor] = None, attention_mask: Optional[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: 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: Tensor, key: Optional[Tensor] = None, key\_ignorance\_mask: Optional[Tensor] = None, attention\_mask: Optional[Tensor] = None, head\_at\_last: bool = True*) → 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: 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: 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: `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: `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 attended, 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: 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 attended, 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: 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: 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\_attention

```
class mwptoolkit.module.Attention.tree_attention.TreeAttention(input_size, hidden_size)
```

Bases: 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: 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: 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: 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, Tensor]

Build dictionary of target matrices.

**Return type**

Dict[str, torch.Tensor]

**Returns**

Dictionary of target values

**\_forward\_single**(*\*\*kwargs*) → Dict[str, Tensor]

Forward computation of a single beam

**Return type**

Dict[str, torch.Tensor]

**Returns**

Dictionary of computed values

**\_init\_weights**(*module: Module*)

Initialize weights

**Parameters**

**module** (*nn.Module*) – Module to be initialized.

**forward**(*text: Optional[Tensor] = None, text\_pad: Optional[Tensor] = None, text\_num: Optional[Tensor] = None, text\_numpad: Optional[Tensor] = None, equation: Optional[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: [DecoderModel](#)

Decoding model that generates expression sequences (Abstract class)

Initiate Equation Builder instance

**Parameters**

**config** (*ModelConfig*) – Configuration of this model

**\_build\_decoder\_context** (*embedding: Tensor, embedding\_pad: Optional[Tensor] = None, text: Optional[Tensor] = None, text\_pad: Optional[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: Tensor, nums: 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: Tensor, mem\_pos: Tensor, nums: Tensor) → Tensor

Build operand embedding a<sub>ij</sub> in the paper.

**Parameters**

- **ids** (torch.Tensor) – LongTensor containing index-type information of operands. (This corresponds to a<sub>ij</sub> 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<sub>{a<sub>ij</sub>}</sub> in the paper)

**Return type**

torch.Tensor

**Returns**

A FloatTensor representing operand embedding vector a<sub>ij</sub> in Equation 3, 4, 5

**\_forward\_single**(text: Optional[Tensor] = None, text\_pad: Optional[Tensor] = None, text\_num: Optional[Tensor] = None, text\_numpad: Optional[Tensor] = None, equation: Optional[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: [ExpressionDecoderModel](#)

The EPT model

Initiate Equation Builder instance

**Parameters**

**config** (*ModelConfig*) – Configuration of this model

**\_build\_attention\_keys**(*num: Tensor*, *mem: Tensor*, *num\_pad: Optional[Tensor] = None*, *mem\_pad: Optional[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\_oprand\_embed**(ids: Tensor, mem\_pos: Tensor, nums: 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[Tensor] = None, text\_pad: Optional[Tensor] = None, text\_num: Optional[Tensor] = None, text\_numpad: Optional[Tensor] = None, equation: Optional[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: [ExpressionDecoderModel](#)

Vanilla Transformer + Expression (The second ablated model)

Initiate Equation Builder instance

**Parameters**

**config** (*ModelConfig*) – Configuration of this model

**\_build\_operand\_embed**(*ids: Tensor, mem\_pos: Tensor, nums: 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[Tensor] = None, text\_pad: Optional[Tensor] = None, text\_num: Optional[Tensor] = None, text\_numpad: Optional[Tensor] = None, equation: Optional[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: LogSoftmax

LogSoftmax layer that can handle infinity values.

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

**dim:** Optional[int]**forward**(tensor: 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: [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: Tensor, embedding\_pad: Optional[Tensor] = None, text: Optional[Tensor] = None, text\_pad: Optional[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: Tensor, nums: 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: Tensor, nums: 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[Tensor] = None, text\_pad: Optional[Tensor] = None, text\_num: Optional[Tensor] = None, text\_numpad: Optional[Tensor] = None, equation: Optional[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: 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: 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: [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: Tensor, nums: 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[Tensor] = None, text\_pad: Optional[Tensor] = None, text\_num: Optional[Tensor] = None, text\_numpad: Optional[Tensor] = None, equation: Optional[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  $\geq dim$ .

**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: ModuleDict, encoded: 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[Embedding, Tensor], tokens: 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  $j \geq i + \text{diagonal}$

**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: 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: 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: 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: 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: *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: 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: 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: 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: 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\_chlds*, *encoder\_outputs*, *num\_pades*, *padding\_hidden*, *seq\_mask*, *nums\_mask*)

**Parameters**

- **node\_stacks** (*list*) – node stacks.
- **left\_chlds** (*list*) – representation of left chlds.
- **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: Module

Seq2tree decoder with Problem aware dynamic encoding

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

**forward**(*node\_stacks*, *left\_chlds*, *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: 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: 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: 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: `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*) → `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: Tensor*) → `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:** `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*) → `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: 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: 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: 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: `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: `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: 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: 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: 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: 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: 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: 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: 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: 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: 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: 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: 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** (*tensor*) – Tensor of shape (dim\_embedding,). The embedding to return when stack is empty.

**add\_variable**(*embedding*)

Tell the stack machine to increase the number of unknown variables by 1.

**Parameters**

**embedding** (*torch.Tensor*) – Tensor of shape (dim\_embedding). Embedding of the unknown variable.

**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\_eq1**(*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  $\geq$  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: 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: 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: 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: 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: 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: 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: 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: `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: `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: `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: `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: `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: 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: 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: `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: `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: `Module`

**Transformer Layer, including**

a multi-head self-attention, an 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: 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: 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: `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: `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: `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: 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: 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.NodeGenerator(hidden_size, op_nums, embedding_size,  
                                                         dropout=0.5)
```

Bases: 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: 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: 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: 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: 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: `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: `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: `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: 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 hypotheses.

**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 sequence, shape: [hyp\_num], torch.Tensor, optional: the chosen states of decoder, shape: [new\_hyp\_num, sequence\_length, hidden\_size] torch.Tensor, optional: the copied output of encoder, shape: [new\_hyp\_num, sequence\_length, hidden\_size] torch.Tensor, optional: the copied 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, False 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  $\geq$  top\_p (nucleus filtering).

**Returns**

the chosen index of token.

**Return type**

*torch.Tensor*



## 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: [\*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 information for 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: [\*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 information for 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: [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 information for 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: [\*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 information for 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: [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 information for 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: *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 information for 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: *GTSTrainer*

multienccdec 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 information for 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: *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 information for 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: [`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 information for 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: [`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 information for 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.SignedTrainer(*config, model, dataloader, evaluator*)

Bases: [SupervisedTrainer](#)

signed trainer, used to implement training, testing, parameter searching for deep-learning model S-aligned.

example of instantiation:

```
>>> trainer = SignedTrainer(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 information for 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: [\*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 information for 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: *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 information for 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: [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 information for 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: [\*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 information for 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: *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**()





## 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: 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: 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: 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', 'NUM', 'NUM', 'NUM', 'NUM', 'NUM', 'NUM', 'NUM', 'NUM', 'NUM', 'NUM',
    'NUM', 'NUM', 'NUM', 'NUM', 'NUM', 'NUM', 'NUM', 'NUM', 'NUM', 'NUM', 'NUM', 'NUM',
    'NUM', 'NUM', 'NUM', 'NUM', 'NUM', 'NUM', 'NUM', 'NUM', 'NUM', 'NUM', 'NUM', 'NUM',
    'NUM', 'NUM', 'NUM', 'NUM', 'NUM', 'NUM', 'NUM', 'NUM', 'NUM', 'NUM', 'NUM', 'NUM',
    'NUM', 'NUM', 'NUM', 'NUM', 'NUM', 'NUM', 'NUM', 'NUM', 'NUM', 'NUM', 'NUM', 'NUM',
    'NUM', 'NUM', 'NUM', 'NUM', 'NUM', 'NUM', 'NUM', 'NUM', 'NUM', 'NUM', 'NUM', 'NUM',
    'NUM', 'NUM', 'NUM', 'NUM', 'NUM', 'NUM', 'NUM', 'NUM', 'NUM', 'NUM', 'NUM', 'NUM',
    'NUM', 'NUM', '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 have 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**

transferred 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_fraction(text_list: List[str]) → List[str]`  
joint fraction number

**Parameters**

**text\_list** – text list.

**Returns**

processed text list.

`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',  
word_lower=False)`

```

mwptoolkit.utils.preprocess_tool.number_transfer.num_transfer_draw(data, mask_type,
                                                                    equ_split_symbol=';',
                                                                    vocab_level='word',
                                                                    word_lower=False)

mwptoolkit.utils.preprocess_tool.number_transfer.num_transfer_hmwp(data, mask_type,
                                                                    equ_split_symbol=';',
                                                                    vocab_level='word',
                                                                    word_lower=False)

mwptoolkit.utils.preprocess_tool.number_transfer.num_transfer_multi(data, mask_type,
                                                                    equ_split_symbol=';',
                                                                    vocab_level='word',
                                                                    word_lower=False)

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',
                                                                    word_lower=False) →
                                                                    Tuple[list, list, int, list]

```

number transfer

#### Parameters

- **datas** (*list*) – dataset.
- **dataset\_name** (*str*) – dataset name.
- **task\_type** (*str*) – [single\_equation | multi\_equation], task type.
- **mask\_type** –
- **min\_generate\_keep** (*int*) – generate number that count greater than the value, will be kept in output symbols.
- **linear\_dataset** (*bool*) –
- **equ\_split\_symbol** (*str*) – equation split symbol, in multiple-equation dataset, symbol to split equations, this symbol will be replaced with special token SpecialTokens.BRG
- **vocab\_level** (*str*) –
- **word\_lower** (*bool*) –

#### Returns

processed datas, generate number list, copy number, unk symbol list.

```

mwptoolkit.utils.preprocess_tool.number_transfer.number_transfer_ape200k(data, mask_type,
                                                                    linear,
                                                                    vocab_level='word',
                                                                    word_lower=False)

mwptoolkit.utils.preprocess_tool.number_transfer.number_transfer_asdiv_a(data, mask_type,
                                                                    linear,
                                                                    vocab_level='word',
                                                                    word_lower=False)

```

```
mwptoolkit.utils.preprocess_tool.number_transfer.number_transfer_math23k(data, mask_type,  
                                                                           linear,  
                                                                           vocab_level='word',  
                                                                           word_lower=False)  
  
mwptoolkit.utils.preprocess_tool.number_transfer.number_transfer_mawps(data, mask_type, linear,  
                                                                           vocab_level='word',  
                                                                           word_lower=False)  
  
mwptoolkit.utils.preprocess_tool.number_transfer.number_transfer_mawps_single(data,  
                                                                           mask_type,  
                                                                           linear, vo-  
                                                                           cab_level='word',  
                                                                           word_lower=False)  
  
mwptoolkit.utils.preprocess_tool.number_transfer.number_transfer_single(data, mask_type,  
                                                                           linear,  
                                                                           vocab_level='word',  
                                                                           word_lower=False)  
  
mwptoolkit.utils.preprocess_tool.number_transfer.number_transfer_svamp(data, mask_type, linear,  
                                                                           vocab_level='word',  
                                                                           word_lower=False)  
  
mwptoolkit.utils.preprocess_tool.number_transfer.seg_and_tag_ape200k(st, nums_fraction, nums)  
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 information to file  
  
mwptoolkit.utils.preprocess_tool.sentence_operator.find_ept_numbers_in_text(text: str, ap-  
                                                                           pend_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 information 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 information 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, lan-  
                                                                              guage,  
                                                                              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



## 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)
```



## **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)
```



## MWPTOOLKIT USAGE:

command line [lookup](#)



## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`





## PYTHON MODULE INDEX

### m

mwptoolkit.config.configuration, 1  
 mwptoolkit.data.dataloader.abstract\_dataloader, 3  
 mwptoolkit.data.dataloader.dataloader\_ept, 4  
 mwptoolkit.data.dataloader.dataloader\_hms, 5  
 mwptoolkit.data.dataloader.dataloader\_multiencdec, 5  
 mwptoolkit.data.dataloader.multi\_equation\_dataloader, 6  
 mwptoolkit.data.dataloader.pretrain\_dataloader, 7  
 mwptoolkit.data.dataloader.single\_equation\_dataloader, 8  
 mwptoolkit.data.dataloader.template\_dataloader, 9  
 mwptoolkit.data.dataset.abstract\_dataset, 10  
 mwptoolkit.data.dataset.dataset\_ept, 12  
 mwptoolkit.data.dataset.dataset\_hms, 13  
 mwptoolkit.data.dataset.dataset\_multiencdec, 14  
 mwptoolkit.data.dataset.multi\_equation\_dataset, 16  
 mwptoolkit.data.dataset.pretrain\_dataset, 17  
 mwptoolkit.data.dataset.single\_equation\_dataset, 19  
 mwptoolkit.data.dataset.template\_dataset, 20  
 mwptoolkit.data.utils, 22  
 mwptoolkit.evaluate.evaluator, 25  
 mwptoolkit.hyper\_search, 157  
 mwptoolkit.loss.abstract\_loss, 31  
 mwptoolkit.loss.binary\_cross\_entropy\_loss, 31  
 mwptoolkit.loss.cross\_entropy\_loss, 32  
 mwptoolkit.loss.masked\_cross\_entropy\_loss, 32  
 mwptoolkit.loss.mse\_loss, 33  
 mwptoolkit.loss.nll\_loss, 33  
 mwptoolkit.loss.smoothed\_cross\_entropy\_loss, 34  
 mwptoolkit.model.Graph2Tree.graph2tree, 61  
 mwptoolkit.model.Graph2Tree.multiencdec, 62  
 mwptoolkit.model.PreTrain.bertgen, 64  
 mwptoolkit.model.PreTrain.gpt2, 65  
 mwptoolkit.model.PreTrain.robertagen, 66  
 mwptoolkit.model.Seq2Seq.dns, 37  
 mwptoolkit.model.Seq2Seq.ept, 39  
 mwptoolkit.model.Seq2Seq.groupatt, 41  
 mwptoolkit.model.Seq2Seq.mathen, 42  
 mwptoolkit.model.Seq2Seq.rnnencdec, 43  
 mwptoolkit.model.Seq2Seq.rnnvae, 45  
 mwptoolkit.model.Seq2Seq.saligned, 46  
 mwptoolkit.model.Seq2Seq.transformer, 47  
 mwptoolkit.model.Seq2Tree.berttd, 49  
 mwptoolkit.model.Seq2Tree.gts, 50  
 mwptoolkit.model.Seq2Tree.mwpbert, 51  
 mwptoolkit.model.Seq2Tree.sausolver, 53  
 mwptoolkit.model.Seq2Tree.treelstm, 54  
 mwptoolkit.model.Seq2Tree.trnn, 56  
 mwptoolkit.model.Seq2Tree.tsn, 57  
 mwptoolkit.module.Attention.group\_attention, 69  
 mwptoolkit.module.Attention.multi\_head\_attention, 70  
 mwptoolkit.module.Attention.self\_attention, 72  
 mwptoolkit.module.Attention.seq\_attention, 73  
 mwptoolkit.module.Attention.tree\_attention, 75  
 mwptoolkit.module.Decoder.ept\_decoder, 75  
 mwptoolkit.module.Decoder.rnn\_decoder, 88  
 mwptoolkit.module.Decoder.transformer\_decoder, 90  
 mwptoolkit.module.Decoder.tree\_decoder, 91  
 mwptoolkit.module.Embedder.basic\_embedder, 95  
 mwptoolkit.module.Embedder.bert\_embedder, 95  
 mwptoolkit.module.Embedder.position\_embedder, 95  
 mwptoolkit.module.Embedder.roberta\_embedder, 98  
 mwptoolkit.module.Encoder.graph\_based\_encoder, 98  
 mwptoolkit.module.Encoder.rnn\_encoder, 100  
 mwptoolkit.module.Encoder.transformer\_encoder, 103  
 mwptoolkit.module.Environment.stack\_machine,

104  
mwptoolkit.module.Graph.gcn, 106  
mwptoolkit.module.Graph.graph\_module, 106  
mwptoolkit.module.Layer.graph\_layers, 108  
mwptoolkit.module.Layer.layers, 109  
mwptoolkit.module.Layer.transformer\_layer,  
110  
mwptoolkit.module.Layer.tree\_layers, 113  
mwptoolkit.module.Strategy.beam\_search, 119  
mwptoolkit.module.Strategy.greedy, 121  
mwptoolkit.module.Strategy.sampling, 121  
mwptoolkit.quick\_start, 159  
mwptoolkit.trainer.abstract\_trainer, 123  
mwptoolkit.trainer.supervised\_trainer, 124  
mwptoolkit.trainer.template\_trainer, 139  
mwptoolkit.utils.data\_structure, 141  
mwptoolkit.utils.enum\_type, 142  
mwptoolkit.utils.logger, 148  
mwptoolkit.utils.preprocess\_tool.dataset\_operator,  
148  
mwptoolkit.utils.preprocess\_tool.equation\_operator,  
148  
mwptoolkit.utils.preprocess\_tool.number\_operator,  
151  
mwptoolkit.utils.preprocess\_tool.number\_transfer,  
152  
mwptoolkit.utils.preprocess\_tool.sentence\_operator,  
154  
mwptoolkit.utils.utils, 155

# INDEX

## Symbols

		method), 21
<code>_build_attention_keys()</code>	(mwp- <code>_build_word_embed()</code>	(mwp- <code>toolkit.module.Decoder.ept_decoder.ExpressionPointerTransformer</code> method), 79
<code>_build_decoder_context()</code>	(mwp- <code>_build_word_embed()</code>	(mwp- <code>toolkit.module.Decoder.ept_decoder.OpDecoderModel</code> method), 84
<code>_build_decoder_context()</code>	(mwp- <code>_build_word_embed()</code>	(mwp- <code>toolkit.module.Decoder.ept_decoder.VanillaOpTransformer</code> method), 85
<code>_build_decoder_context()</code>	(mwp- <code>_compute_expression_by_postfix_multi()</code>	(mwp- <code>toolkit.evaluate.evaluator.InfixEvaluator</code> method), 25
<code>_build_decoder_input()</code>	(mwp- <code>_compute_expression_by_postfix_multi()</code>	(mwp- <code>toolkit.evaluate.evaluator.MultiWayTreeEvaluator</code> method), 28
<code>_build_decoder_input()</code>	(mwp- <code>_convert_config_dict()</code>	(mwp- <code>toolkit.config.configuration.Config</code> method), 1
<code>_build_operand_embed()</code>	(mwp- <code>_eval_batch()</code>	(mwp- <code>toolkit.trainer.supervised_trainer.EPTTrainer</code> method), 125
<code>_build_operand_embed()</code>	(mwp- <code>_forward()</code>	(mwp- <code>toolkit.module.Embedder.position_embedder.EPTPositionEmbedder</code> method), 96
<code>_build_operand_embed()</code>	(mwp- <code>_forward_single()</code>	(mwp- <code>toolkit.module.Decoder.ept_decoder.DecoderModel</code> method), 76
<code>_build_operand_embed()</code>	(mwp- <code>_forward_single()</code>	(mwp- <code>toolkit.module.Decoder.ept_decoder.ExpressionTransformer</code> method), 81
<code>_build_symbol()</code>	(mwp- <code>_forward_single()</code>	(mwp- <code>toolkit.module.Decoder.ept_decoder.ExpressionDecoderModel</code> method), 78
<code>_build_target_dict()</code>	(mwp- <code>_forward_single()</code>	(mwp- <code>toolkit.module.Decoder.ept_decoder.ExpressionPointerTransformer</code> method), 80
<code>_build_target_dict()</code>	(mwp- <code>_forward_single()</code>	(mwp- <code>toolkit.module.Decoder.ept_decoder.ExpressionTransformer</code> method), 82
<code>_build_target_dict()</code>	(mwp- <code>_forward_single()</code>	(mwp- <code>toolkit.module.Decoder.ept_decoder.OpDecoderModel</code> method), 84
<code>_build_target_dict()</code>	(mwp- <code>_forward_single()</code>	(mwp- <code>toolkit.module.Decoder.ept_decoder.VanillaOpTransformer</code> method), 86
<code>_build_target_dict()</code>	(mwp- <code>_init_weights()</code>	(mwp- <code>toolkit.module.Decoder.ept_decoder.DecoderModel</code> method), 76
<code>_build_template_symbol()</code>	(mwp- <code>_init_weights()</code>	(mwp- <code>toolkit.data.dataset.template_dataset.TemplateDataset</code> method), 76

`_load_cmd_line()` (mwptoolkit.config.configuration.Config method), 2  
`_load_dataset()` (mwptoolkit.data.dataset.abstract\_dataset.AbstractDataset method), 11  
`_load_fold_dataset()` (mwptoolkit.data.dataset.abstract\_dataset.AbstractDataset method), 11  
`_normalize_gradients()` (mwptoolkit.trainer.supervised\_trainer.EPTTrainer method), 125  
`_preprocess()` (mwptoolkit.data.dataset.template\_dataset.TemplateDataset method), 21

## A

`AbstractDataLoader` (class in mwptoolkit.data.data\_loader.abstract\_data\_loader), 3  
`AbstractDataset` (class in mwptoolkit.data.dataset.abstract\_dataset), 10  
`AbstractEvaluator` (class in mwptoolkit.evaluate.evaluator), 25  
`AbstractLoss` (class in mwptoolkit.loss.abstract\_loss), 31  
`AbstractTrainer` (class in mwptoolkit.trainer.abstract\_trainer), 123  
`AbstractTree` (class in mwptoolkit.utils.data\_structure), 141  
`add_child()` (mwptoolkit.utils.data\_structure.Tree method), 142  
`add_left_node()` (mwptoolkit.utils.data\_structure.DependencyNode method), 141  
`add_norm()` (mwptoolkit.loss.binary\_cross\_entropy\_loss.BinaryCrossEntropyLoss method), 31  
`add_right_node()` (mwptoolkit.utils.data\_structure.DependencyNode method), 141  
`add_variable()` (mwptoolkit.module.Environment.stack\_machine.StackMachine method), 105  
`adjust_equ()` (mwptoolkit.trainer.supervised\_trainer.SalignedTrainer method), 133  
`alg514` (mwptoolkit.utils.enum\_type.DatasetName attribute), 142  
`alphabet` (mwptoolkit.utils.enum\_type.MaskSymbol attribute), 146  
`alphabet` (mwptoolkit.utils.enum\_type.NumMask attribute), 146  
`ans_module_calculate_loss()` (mwptoolkit.model.Seq2Tree.trnn.TRNN method), 56  
`ans_module_forward()` (mwptoolkit.model.Seq2Tree.trnn.TRNN method), 56  
`ape200k` (mwptoolkit.utils.enum\_type.DatasetName attribute), 142  
`apply_across_dim()` (in module mwptoolkit.module.Decoder.ept\_decoder), 86  
`apply_embed_only()` (mwptoolkit.module.Environment.stack\_machine.StackMachine method), 105  
`apply_eq1()` (mwptoolkit.module.Environment.stack\_machine.StackMachine method), 105  
`apply_module_dict()` (in module mwptoolkit.module.Decoder.ept\_decoder), 87  
`ARG_CON` (mwptoolkit.utils.enum\_type.EPT attribute), 143  
`ARG_CON_ID` (mwptoolkit.utils.enum\_type.EPT attribute), 143  
`ARG_MEM` (mwptoolkit.utils.enum\_type.EPT attribute), 143  
`ARG_MEM_ID` (mwptoolkit.utils.enum\_type.EPT attribute), 143  
`ARG_NUM` (mwptoolkit.utils.enum\_type.EPT attribute), 143  
`ARG_NUM_ID` (mwptoolkit.utils.enum\_type.EPT attribute), 143  
`ARG_TOKENS` (mwptoolkit.utils.enum\_type.EPT attribute), 143  
`ARG_UNK` (mwptoolkit.utils.enum\_type.EPT attribute), 143  
`ARG_UNK_ID` (mwptoolkit.utils.enum\_type.EPT attribute), 143  
`ARITY_MAP` (mwptoolkit.utils.enum\_type.EPT attribute), 143  
`asdiv_a` (mwptoolkit.utils.enum\_type.DatasetName attribute), 142  
`Attention` (class in mwptoolkit.module.Attention.seq\_attention), 73  
`attention()` (in module mwptoolkit.module.Attention.group\_attention), 69  
`AttentionalRNND decoder` (class in mwptoolkit.module.Decoder.rnn\_decoder), 88  
`attn_decoder_forward()` (mwptoolkit.model.Graph2Tree.multienccdec.MultiEncDec method), 62  
`AveragePooling` (class in mwptoolkit.module.Decoder.ept\_decoder), 75

## B

`b_normal()` (mwptoolkit.module.Graph.graph\_module.Graph\_Module method), 106  
`backward()` (mwptoolkit.loss.abstract\_loss.AbstractLoss method), 31

BasicEmbedder (class in mwp-toolkit.module.Embedder.basic_embedder), 95	build_graph() (mwp-toolkit.model.Graph2Tree.graph2tree.Graph2Tree method), 61
BasicRNNDecoder (class in mwp-toolkit.module.Decoder.rnn_decoder), 88	build_graph() (mwp-toolkit.model.Seq2Tree.tsn.TSN method), 57
BasicRNNEncoder (class in mwp-toolkit.module.Encoder.rnn_encoder), 100	build_pos_to_file_with_pyltp() (mwp-toolkit.data.dataset.dataset_multiencdec.DatasetMultiEncDec method), 15
Beam (class in mwp-toolkit.module.Strategy.beam_search), 119	build_pos_to_file_with_stanza() (mwp-toolkit.data.dataset.dataset_multiencdec.DatasetMultiEncDec method), 15
BeamSearchHypothesis (class in mwp-toolkit.module.Strategy.beam_search), 120	<b>C</b>
BeamNode (class in mwp-toolkit.module.Strategy.beam_search), 119	calculate_loss() (mwp-toolkit.model.Graph2Tree.graph2tree.Graph2Tree method), 61
before_trigonometric() (mwp-toolkit.module.Embedder.position_embedder.EPTPositionalEncoding method), 96	calculate_loss() (mwp-toolkit.model.Graph2Tree.multiencdec.MultiEncDec method), 62
BertEmbedder (class in mwp-toolkit.module.Embedder.bert_embedder), 95	calculate_loss() (mwp-toolkit.model.PreTrain.bertgen.BERTGen method), 64
BertEncoder (class in mwp-toolkit.module.Encoder.transformer_encoder), 103	calculate_loss() (mwp-toolkit.model.PreTrain.gpt2.GPT2 method), 65
BERTGen (class in mwp-toolkit.model.PreTrain.bertgen), 64	calculate_loss() (mwp-toolkit.model.PreTrain.robertagen.RobertaGen method), 66
BertTD (class in mwp-toolkit.model.Seq2Tree.berttd), 49	calculate_loss() (mwp-toolkit.model.Seq2Seq.dns.DNS method), 37
BertTDTrainer (class in mwp-toolkit.trainer.supervised_trainer), 124	calculate_loss() (mwp-toolkit.model.Seq2Seq.ept.EPT method), 39
bi_combine() (mwp-toolkit.module.Encoder.rnn_encoder.HWCPEncoder method), 101	calculate_loss() (mwp-toolkit.model.Seq2Seq.groupatt.GroupATT method), 41
BinaryCrossEntropyLoss (class in mwp-toolkit.loss.binary_cross_entropy_loss), 31	calculate_loss() (mwp-toolkit.model.Seq2Seq.mathen.MathEN method), 42
BinaryTree (class in mwp-toolkit.utils.data_structure), 141	calculate_loss() (mwp-toolkit.model.Seq2Seq.rnnencdec.RNNEncDec method), 43
BRG_TOKEN (mwp-toolkit.utils.enum_type.SpecialTokens attribute), 147	calculate_loss() (mwp-toolkit.model.Seq2Seq.rnnvae.RNNVAE method), 45
build_batch_for_predict() (mwp-toolkit.data.dataloader.dataloader_ept.DataLoaderEPT method), 4	calculate_loss() (mwp-toolkit.model.Seq2Seq.saligned.Saligned method), 46
build_batch_for_predict() (mwp-toolkit.data.dataloader.dataloader_hms.DataLoaderHMS method), 5	calculate_loss() (mwp-toolkit.model.Seq2Seq.transformer.Transformer method), 47
build_batch_for_predict() (mwp-toolkit.data.dataloader.dataloader_multiencdec.DataLoaderMultiEncDec method), 6	calculate_loss() (mwp-toolkit.model.Seq2Tree.berttd.BertTD method), 49
build_batch_for_predict() (mwp-toolkit.data.dataloader.multi_equation_dataloader.MultiEquationDataLoader method), 6	
build_batch_for_predict() (mwp-toolkit.data.dataloader.pretrain_dataloader.PretrainDataLoader method), 7	
build_batch_for_predict() (mwp-toolkit.data.dataloader.single_equation_dataloader.SingleEquationDataLoader method), 8	

49			<i>toolkit.model.Seq2Seq.rnnencdec.RNNEncDec</i>
<code>calculate_loss()</code>	(mwp- <i>toolkit.model.Seq2Tree.gts.GTS</i> method),	<code>convert_idx2symbol()</code>	(mwp- <i>toolkit.model.Seq2Seq.rnnvae.RNNVAE</i> method), 43
50			
<code>calculate_loss()</code>	(mwp- <i>toolkit.model.Seq2Tree.mwpbert.MWPBert</i> method), 51	<code>convert_idx2symbol()</code>	(mwp- <i>toolkit.model.Seq2Seq.saligned.Saligned</i> method), 45
<code>calculate_loss()</code>	(mwp- <i>toolkit.model.Seq2Tree.sausolver.SAUSolver</i> method), 53	<code>convert_idx2symbol()</code>	(mwp- <i>toolkit.model.Seq2Seq.transformer.Transformer</i> method), 46
<code>calculate_loss()</code>	(mwp- <i>toolkit.model.Seq2Tree.treelstm.TreeLSTM</i> method), 54	<code>convert_idx2symbol()</code>	(mwp- <i>toolkit.model.Seq2Tree.berttd.BertTD</i> method), 49
<code>calculate_loss()</code>	(mwp- <i>toolkit.model.Seq2Tree.trnn.TRNN</i> method),	<code>convert_idx2symbol()</code>	(mwp- <i>toolkit.model.Seq2Tree.gts.GTS</i> method), 50
<code>clause_level_forward()</code>	(mwp- <i>toolkit.module.Encoder.rnn_encoder.HWCPEncoder</i> method), 101	<code>convert_idx2symbol()</code>	(mwp- <i>toolkit.model.Seq2Tree.mwpbert.MWPBert</i> method), 52
<code>clones()</code> (in module <i>mwp toolkit.utils.utils</i> ), 155		<code>convert_idx2symbol()</code>	(mwp- <i>toolkit.model.Seq2Tree.sausolver.SAUSolver</i> method), 53
<code>CON_PREFIX</code> ( <i>mwp toolkit.utils.enum_type.EPT</i> attribute), 143		<code>convert_idx2symbol()</code>	(mwp- <i>toolkit.model.Seq2Tree.treelstm.TreeLSTM</i> method), 54
<code>Config</code> (class in <i>mwp toolkit.config.configuration</i> ), 1		<code>convert_idx2symbol()</code>	(mwp- <i>toolkit.model.Seq2Tree.trnn.TRNN</i> method), 56
<code>constant_number()</code> (in module <i>mwp- toolkit.utils.preprocess_tool.number_operator</i> ), 151		<code>convert_idx2symbol()</code>	(mwp- <i>toolkit.model.Seq2Tree.tsn.TSN</i> method), 57
<code>constant_word_embedding</code> (mwp- <i>toolkit.module.Decoder.ept_decoder.ExpressionPointerTransformer</i> attribute), 81		<code>convert_idx2symbol1()</code>	(mwp- <i>toolkit.model.Graph2Tree.multienccdec.MultiEncDec</i> method), 62
<code>convert_idx2symbol()</code> (mwp- <i>toolkit.model.Graph2Tree.graph2tree.Graph2Tree</i> method), 61		<code>convert_idx2symbol2()</code>	(mwp- <i>toolkit.model.Graph2Tree.multienccdec.MultiEncDec</i> method), 62
<code>convert_idx2symbol()</code> (mwp- <i>toolkit.model.PreTrain.bertgen.BERTGen</i> method), 64		<code>convert_idx_2_symbol()</code>	(mwp- <i>toolkit.data.dataloader.abstract_dataloader.AbstractDataLoader</i> method), 3
<code>convert_idx2symbol()</code> (mwp- <i>toolkit.model.PreTrain.gpt2.GPT2</i> method), 65		<code>convert_idx_2_word()</code>	(mwp- <i>toolkit.data.dataloader.abstract_dataloader.AbstractDataLoader</i> method), 3
<code>convert_idx2symbol()</code> (mwp- <i>toolkit.model.PreTrain.robertagen.RobertaGen</i> method), 66		<code>convert_in_idx_2_out_idx()</code>	(mwp- <i>toolkit.model.Seq2Seq.dns.DNS</i> method), 37
<code>convert_idx2symbol()</code> (mwp- <i>toolkit.model.Seq2Seq.dns.DNS</i> method), 37		<code>convert_in_idx_2_out_idx()</code>	(mwp- <i>toolkit.model.Seq2Seq.groupatt.GroupATT</i> method), 41
<code>convert_idx2symbol()</code> (mwp- <i>toolkit.model.Seq2Seq.ept.EPT</i> method), 39		<code>convert_in_idx_2_out_idx()</code>	(mwp- <i>toolkit.model.Seq2Seq.mathen.MathEN</i> method), 42
<code>convert_idx2symbol()</code> (mwp- <i>toolkit.model.Seq2Seq.groupatt.GroupATT</i> method), 41		<code>convert_in_idx_2_out_idx()</code>	(mwp- <i>toolkit.model.Seq2Seq.mathen.MathEN</i> method), 42
<code>convert_idx2symbol()</code> (mwp- <i>toolkit.model.Seq2Seq.mathen.MathEN</i> method), 42		<code>convert_in_idx_2_out_idx()</code>	(mwp- <i>toolkit.model.Seq2Seq.mathen.MathEN</i> method), 42
<code>convert_idx2symbol()</code> (mwp- <i>toolkit.model.Seq2Seq.mathen.MathEN</i> method), 42		<code>convert_in_idx_2_out_idx()</code>	(mwp- <i>toolkit.model.Seq2Seq.mathen.MathEN</i> method), 42



`toolkit.model.Seq2Seq.rnnencdec.RNNEncDec`  
`method`), 44  
`convert_in_idx_2_out_idx()` (*mwp-*  
`toolkit.model.Seq2Seq.rnnvae.RNNVAE  
method), 45  
convert_in_idx_2_out_idx() (mwp-  
toolkit.model.Seq2Seq.transformer.Transformer  
method), 48  
convert_in_idx_2_temp_idx() (mwp-  
toolkit.model.Seq2Tree.trnn.TRNN method),  
46  
convert_mask_num() (mwp-  
toolkit.model.Seq2Seq.saligned.Saligned  
method), 46  
convert_out_idx_2_in_idx() (mwp-  
toolkit.model.Seq2Seq.dns.DNS method),  
37  
convert_out_idx_2_in_idx() (mwp-  
toolkit.model.Seq2Seq.groupatt.GroupATT  
method), 41  
convert_out_idx_2_in_idx() (mwp-  
toolkit.model.Seq2Seq.mathen.MathEN  
method), 42  
convert_out_idx_2_in_idx() (mwp-  
toolkit.model.Seq2Seq.rnnencdec.RNNEncDec  
method), 44  
convert_out_idx_2_in_idx() (mwp-  
toolkit.model.Seq2Seq.rnnvae.RNNVAE  
method), 45  
convert_out_idx_2_in_idx() (mwp-  
toolkit.model.Seq2Seq.transformer.Transformer  
method), 48  
convert_symbol_2_idx() (mwp-  
toolkit.data.dataloader.abstract_dataloader.AbstractDataLoader  
method), 3  
convert_temp_idx2symbol() (mwp-  
toolkit.model.Seq2Tree.trnn.TRNN method),  
56  
convert_temp_idx_2_in_idx() (mwp-  
toolkit.model.Seq2Tree.trnn.TRNN method),  
56  
convert_word_2_idx() (mwp-  
toolkit.data.dataloader.abstract_dataloader.AbstractDataLoader  
method), 4  
copy() (mwptoolkit.module.Strategy.beam_search.BeamNode  
method), 119  
copy_list() (in module mwptoolkit.utils.utils), 155  
copy_list() (mwptoolkit.model.Seq2Tree.treelstm.TreeLSTM  
method), 55  
cosine_loss() (in module mwp-  
toolkit.model.Seq2Tree.tsn), 60  
cosine_sim() (in module mwp-  
toolkit.model.Seq2Tree.tsn), 60  
create_dataloader() (in module mwp-  
toolkit.data.utils), 22  
create_dataset() (in module mwptoolkit.data.utils),  
22  
cross_validation_load() (mwp-  
toolkit.data.dataset.abstract_dataset.AbstractDataset  
method), 11  
CrossEntropyLoss (class in mwp-  
toolkit.loss.cross_entropy_loss), 32`

## D

`DataLoaderEPT` (*class in mwp-*  
`toolkit.data.dataloader.dataloader_ept`),  
4  
`DataLoaderHMS` (*class in mwp-*  
`toolkit.data.dataloader.dataloader_hms`),  
5  
`DataLoaderMultiEncDec` (*class in mwp-*  
`toolkit.data.dataloader.dataloader_multiencdec`),  
5  
`dataset_load()` (*mwp-*  
`toolkit.data.dataset.abstract_dataset.AbstractDataset  
method), 11  
DatasetEPT (class in mwp-  
toolkit.data.dataset.dataset_ept), 12  
DatasetHMS (class in mwp-  
toolkit.data.dataset.dataset_hms), 13  
DatasetLanguage (class in mwp-  
toolkit.utils.enum_type), 142  
DatasetMultiEncDec (class in mwp-  
toolkit.data.dataset.dataset_multiencdec),  
14  
DatasetName (class in mwptoolkit.utils.enum_type), 142  
DatasetType (class in mwptoolkit.utils.enum_type), 143  
DeepLstm (class in mwp-  
toolkit.module.Layer.tree_layers), 113  
decode() (mwptoolkit.model.PreTrain.bertgen.BERTGen  
method), 64  
decode() (mwptoolkit.model.PreTrain.robertagen.RobertaGen  
method), 66  
decode() (mwptoolkit.model.Seq2Seq.dns.DNS method),  
37  
decode() (mwptoolkit.model.Seq2Seq.ept.EPT method),  
44  
decode() (mwptoolkit.model.Seq2Seq.groupatt.GroupATT  
method), 41  
decode() (mwptoolkit.model.Seq2Seq.mathen.MathEN  
method), 42  
decode() (mwptoolkit.model.Seq2Seq.rnnencdec.RNNEncDec  
method), 44  
decode() (mwptoolkit.model.Seq2Seq.rnnvae.RNNVAE  
method), 45  
decode() (mwptoolkit.model.Seq2Seq.transformer.Transformer  
method), 48`

`decode_()` (*mwptoolkit.model.PreTrain.bertgen.BERTGen* `method`), 64  
`decode_()` (*mwptoolkit.model.PreTrain.gpt2.GPT2* `method`), 65  
`decode_()` (*mwptoolkit.model.PreTrain.robertagen.RobertaGen* `method`), 66  
`decoder_forward()` (*mwptoolkit.model.Graph2Tree.graph2tree.Graph2Tree* `method`), 61  
`decoder_forward()` (*mwptoolkit.model.Graph2Tree.mutiencdec.MultiEncDec* `method`), 62  
`decoder_forward()` (*mwptoolkit.model.PreTrain.bertgen.BERTGen* `method`), 64  
`decoder_forward()` (*mwptoolkit.model.PreTrain.gpt2.GPT2* `method`), 65  
`decoder_forward()` (*mwptoolkit.model.PreTrain.robertagen.RobertaGen* `method`), 66  
`decoder_forward()` (*mwptoolkit.model.Seq2Seq.dns.DNS* `method`), 37  
`decoder_forward()` (*mwptoolkit.model.Seq2Seq.ept.EPT* `method`), 39  
`decoder_forward()` (*mwptoolkit.model.Seq2Seq.groupatt.GroupATT* `method`), 41  
`decoder_forward()` (*mwptoolkit.model.Seq2Seq.mathen.MathEN* `method`), 42  
`decoder_forward()` (*mwptoolkit.model.Seq2Seq.rnnencdec.RNNEncDec* `method`), 44  
`decoder_forward()` (*mwptoolkit.model.Seq2Seq.rnnvae.RNNVAE* `method`), 45  
`decoder_forward()` (*mwptoolkit.model.Seq2Seq.saligned.Saligned* `method`), 46  
`decoder_forward()` (*mwptoolkit.model.Seq2Seq.transformer.Transformer* `method`), 48  
`decoder_forward()` (*mwptoolkit.model.Seq2Tree.berttd.BertTD* `method`), 49  
`decoder_forward()` (*mwptoolkit.model.Seq2Tree.gts.GTS* `method`), 50  
`decoder_forward()` (*mwptoolkit.model.Seq2Tree.mwpbert.MWPBert* `method`), 52  
`decoder_forward()` (*mwptoolkit.model.Seq2Tree.sausolver.SAUSolver* `method`), 53  
`decoder_forward()` (*mwptoolkit.model.Seq2Tree.treelstm.TreeLSTM* `method`), 55  
`DecoderModel` (class in *mwptoolkit.module.Decoder.ept\_decoder*), 76  
`DecomposeModel` (class in *mwptoolkit.module.Layer.tree\_layers*), 114  
`dependency_encode()` (*mwptoolkit.module.Encoder.rnn\_encoder.HWCPEncoder* `method`), 101  
`DependencyNode` (class in *mwptoolkit.utils.data\_structure*), 141  
`DependencyTree` (class in *mwptoolkit.utils.data\_structure*), 141  
`deprel_tree_to_file()` (in module *mwptoolkit.utils.preprocess\_tool.sentence\_operator*), 154  
`device` (*mwptoolkit.module.Embedder.position\_embedder.EPTPositionalEncoder* property), 96  
`dim` (*mwptoolkit.module.Decoder.ept\_decoder.LogSoftmax* attribute), 83  
`DisPositionalEncoding` (class in *mwptoolkit.module.Embedder.position\_embedder*), 95  
`DNS` (class in *mwptoolkit.model.Seq2Seq.dns*), 37  
`DQN` (class in *mwptoolkit.module.Layer.tree\_layers*), 113  
`draw` (*mwptoolkit.utils.enum\_type.DatasetName* attribute), 142  

## E

`embed_to_hidden` (*mwptoolkit.module.Decoder.ept\_decoder.ExpressionDecoderModel* attribute), 79  
`embedding_dim` (*mwptoolkit.module.Embedder.position\_embedder.EPTPositionalEncoder* attribute), 96  
`en` (*mwptoolkit.utils.enum\_type.DatasetLanguage* attribute), 142  
`en_rule1_process()` (*mwptoolkit.data.dataset.abstract\_dataset.AbstractDataset* `method`), 11  
`EN_rule1_stat()` (in module *mwptoolkit.utils.preprocess\_tool.equation\_operator*), 148  
`EN_rule2()` (in module *mwptoolkit.utils.preprocess\_tool.equation\_operator*), 148  
`en_rule2_process()` (*mwptoolkit.data.dataset.abstract\_dataset.AbstractDataset* `method`), 11



`encode_()` (`mwptoolkit.model.PreTrain.gpt2.GPT2` `method`), 65  
`encoder_forward()` (`mwptoolkit.model.Graph2Tree.graph2tree.Graph2Tree` `method`), 61  
`encoder_forward()` (`mwptoolkit.model.Graph2Tree.multienccdec.MultiEncDec` `method`), 63  
`encoder_forward()` (`mwptoolkit.model.PreTrain.bertgen.BERTGen` `method`), 64  
`encoder_forward()` (`mwptoolkit.model.PreTrain.robertagen.RobertaGen` `method`), 66  
`encoder_forward()` (`mwptoolkit.model.Seq2Seq.dns.DNS` `method`), 37  
`encoder_forward()` (`mwptoolkit.model.Seq2Seq.ept.EPT` `method`), 39  
`encoder_forward()` (`mwptoolkit.model.Seq2Seq.groupatt.GroupATT` `method`), 41  
`encoder_forward()` (`mwptoolkit.model.Seq2Seq.mathen.MathEN` `method`), 42  
`encoder_forward()` (`mwptoolkit.model.Seq2Seq.rnnencdec.RNNEncDec` `method`), 44  
`encoder_forward()` (`mwptoolkit.model.Seq2Seq.rnnvae.RNNVAE` `method`), 45  
`encoder_forward()` (`mwptoolkit.model.Seq2Seq.saligned.Saligned` `method`), 46  
`encoder_forward()` (`mwptoolkit.model.Seq2Seq.transformer.Transformer` `method`), 48  
`encoder_forward()` (`mwptoolkit.model.Seq2Tree.berttd.BertTD` `method`), 49  
`encoder_forward()` (`mwptoolkit.model.Seq2Tree.gts.GTS` `method`), 50  
`encoder_forward()` (`mwptoolkit.model.Seq2Tree.mwpbert.MWPBert` `method`), 52  
`encoder_forward()` (`mwptoolkit.model.Seq2Tree.sausolver.SAUSolver` `method`), 53  
`encoder_forward()` (`mwptoolkit.model.Seq2Tree.treelstm.TreeLSTM` `method`), 55  
`english_word_2_num()` (`in module mwptoolkit.utils.preprocess_tool.number_operator`), 151  
`EOS_TOKEN` (`mwptoolkit.utils.enum_type.SpecialTokens` `attribute`), 147  
`EPT` (`class in mwptoolkit.model.Seq2Seq.ept`), 39  
`EPT` (`class in mwptoolkit.utils.enum_type`), 143  
`Dept_preprocess()` (`in module mwptoolkit.utils.preprocess_tool.dataset_operator`), 148  
`EPTMultiHeadAttention` (`class in mwptoolkit.module.Attention.multi_head_attention`), 70  
`EPTMultiHeadAttentionWeights` (`class in mwptoolkit.module.Attention.multi_head_attention`), 71  
`EPTPositionalEncoding` (`class in mwptoolkit.module.Embedder.position_embedder`), 96  
`EPTTrainer` (`class in mwptoolkit.trainer.supervised_trainer`), 124  
`EPTTransformerLayer` (`class in mwptoolkit.module.Layer.transformer_layer`), 110  
`equ2tree()` (`mwptoolkit.utils.data_structure.AbstractTree` `method`), 141  
`equ2tree()` (`mwptoolkit.utils.data_structure.BinaryTree` `method`), 141  
`equ2tree()` (`mwptoolkit.utils.data_structure.GoldTree` `method`), 141  
`equ2tree_()` (`mwptoolkit.utils.data_structure.BinaryTree` `method`), 141  
`eval_batch()` (`mwptoolkit.loss.abstract_loss.AbstractLoss` `method`), 31  
`eval_batch()` (`mwptoolkit.loss.binary_cross_entropy_loss.BinaryCrossEntropyLoss` `method`), 31  
`eval_batch()` (`mwptoolkit.loss.cross_entropy_loss.CrossEntropyLoss` `method`), 32  
`eval_batch()` (`mwptoolkit.loss.masked_cross_entropy_loss.MaskedCrossEntropyLoss` `method`), 32  
`eval_batch()` (`mwptoolkit.loss.mse_loss.MSELoss` `method`), 33  
`eval_batch()` (`mwptoolkit.loss.nll_loss.NLLLoss` `method`), 33  
`eval_batch()` (`mwptoolkit.loss.smoothed_cross_entropy_loss.SmoothCrossEntropyLoss` `method`), 34  
`eval_source()` (`mwptoolkit.evaluate.evaluator.PostfixEvaluator` `method`), 28  
`eval_source()` (`mwptoolkit.evaluate.evaluator.PrefixEvaluator` `method`), 29  
`evaluate()` (`mwptoolkit.trainer.abstract_trainer.AbstractTrainer` `method`), 123  
`evaluate()` (`mwptoolkit.trainer.supervised_trainer.EPTTrainer` `method`), 123

- method), 125
- evaluate() (mwptoolkit.trainer.supervised\_trainer.GTSTrainer method), 127
- evaluate() (mwptoolkit.trainer.supervised\_trainer.SalignedTrainer method), 134
- evaluate() (mwptoolkit.trainer.supervised\_trainer.SupervisedTrainer method), 138
- evaluate() (mwptoolkit.trainer.supervised\_trainer.TreeLSTMTrainer method), 136
- evaluate() (mwptoolkit.trainer.supervised\_trainer.TRNNTTrainer method), 137
- evaluate() (mwptoolkit.trainer.template\_trainer.TemplateTrainer method), 139
- evaluate\_student() (mwptoolkit.trainer.supervised\_trainer.TSNTrainer method), 137
- evaluate\_teacher() (mwptoolkit.trainer.supervised\_trainer.TSNTrainer method), 137
- evaluate\_tree() (mwptoolkit.model.Seq2Tree.sausolver.SAUSolver method), 53
- ExpressionDecoderModel (class in mwptoolkit.module.Decoder.ept\_decoder), 77
- ExpressionPointerTransformer (class in mwptoolkit.module.Decoder.ept\_decoder), 79
- ExpressionTransformer (class in mwptoolkit.module.Decoder.ept\_decoder), 81
- extra\_repr() (mwptoolkit.module.Decoder.ept\_decoder.AbstractDecoder method), 76
- extra\_repr() (mwptoolkit.module.Decoder.ept\_decoder.Sausolver method), 85
- F**
- FIELD\_EXPR\_GEN (mwptoolkit.utils.enum\_type.EPT attribute), 143
- FIELD\_EXPR\_PTR (mwptoolkit.utils.enum\_type.EPT attribute), 143
- FIELD\_OP\_GEN (mwptoolkit.utils.enum\_type.EPT attribute), 143
- filter\_END() (mwptoolkit.model.Seq2Seq.dns.DNS method), 37
- filter\_op() (mwptoolkit.model.Seq2Seq.dns.DNS method), 37
- find\_ept\_numbers\_in\_text() (in module mwptoolkit.utils.preprocess\_tool.sentence\_operator), 154
- fit() (mwptoolkit.trainer.abstract\_trainer.AbstractTrainer method), 123
- fit() (mwptoolkit.trainer.supervised\_trainer.EPTTrainer method), 126
- fit() (mwptoolkit.trainer.supervised\_trainer.GTSTrainer method), 127
- fit() (mwptoolkit.trainer.supervised\_trainer.SalignedTrainer method), 133
- fit() (mwptoolkit.trainer.supervised\_trainer.SupervisedTrainer method), 134
- fit() (mwptoolkit.trainer.supervised\_trainer.TreeLSTMTrainer method), 136
- fit() (mwptoolkit.trainer.supervised\_trainer.TRNNTTrainer method), 137
- fit() (mwptoolkit.trainer.template\_trainer.TemplateTrainer method), 139
- fix\_process() (mwptoolkit.data.dataset.abstract\_dataset.AbstractDataset method), 11
- FixType (class in mwptoolkit.utils.enum\_type), 146
- FOLLOWING\_ZERO\_PATTERN (mwptoolkit.utils.enum\_type.EPT attribute), 143
- FORMAT\_MEM (mwptoolkit.utils.enum\_type.EPT attribute), 143
- FORMAT\_NUM (mwptoolkit.utils.enum\_type.EPT attribute), 143
- FORMAT\_VAR (mwptoolkit.utils.enum\_type.EPT attribute), 143
- forward() (mwptoolkit.loss.smoothed\_cross\_entropy\_loss.SmoothedCrossEntropyLoss method), 34
- forward() (mwptoolkit.model.Graph2Tree.graph2tree.Graph2Tree method), 61
- forward() (mwptoolkit.model.Graph2Tree.multienccdec.MultiEncDec method), 63
- forward() (mwptoolkit.model.PreTrain.bertgen.BERTGen method), 64
- forward() (mwptoolkit.model.PreTrain.gpt2.GPT2 method), 65
- forward() (mwptoolkit.model.PreTrain.robertagen.RobertaGen method), 67
- forward() (mwptoolkit.model.Seq2Seq.dns.DNS method), 37
- forward() (mwptoolkit.model.Seq2Seq.ept.EPT method), 39
- forward() (mwptoolkit.model.Seq2Seq.groupatt.GroupATT method), 41
- forward() (mwptoolkit.model.Seq2Seq.mathen.MathEN method), 42
- forward() (mwptoolkit.model.Seq2Seq.rnnencdec.RNNEncDec method), 44
- forward() (mwptoolkit.model.Seq2Seq.rnnvae.RNNVAE method), 45
- forward() (mwptoolkit.model.Seq2Seq.saligned.Saligned method), 46
- forward() (mwptoolkit.model.Seq2Seq.transformer.Transformer method), 48
- forward() (mwptoolkit.model.Seq2Tree.berttd.BertTD method), 49

`forward()` (`mwptoolkit.model.Seq2Tree.gts.GTS` `forward()` (`mwptoolkit.module.Decoder.tree_decoder.PredictModel`  
`method`), 50 `method`), 93  
`forward()` (`mwptoolkit.model.Seq2Tree.mwpbert.MWPBer` `forward()` (`mwptoolkit.module.Decoder.tree_decoder.RNNBasedTreeDeco`  
`method`), 52 `method`), 93  
`forward()` (`mwptoolkit.model.Seq2Tree.sausolver.SAUSolve` `forward()` (`mwptoolkit.module.Decoder.tree_decoder.SARTreeDecoder`  
`method`), 53 `method`), 93  
`forward()` (`mwptoolkit.model.Seq2Tree.treelstm.TreeLSTM` `forward()` (`mwptoolkit.module.Decoder.tree_decoder.TreeDecoder`  
`method`), 55 `method`), 94  
`forward()` (`mwptoolkit.model.Seq2Tree.trnn.TRNN` `forward()` (`mwptoolkit.module.Embedder.basic_embedder.BasicEmbedder`  
`method`), 56 `method`), 95  
`forward()` (`mwptoolkit.model.Seq2Tree.tsn.TSN` `forward()` (`mwptoolkit.module.Embedder.bert_embedder.BertEmbedder`  
`method`), 58 `method`), 95  
`forward()` (`mwptoolkit.module.Attention.group_attention.GroupAttention` `forward()` (`mwptoolkit.module.Embedder.position_embedder.DisPositiona`  
`method`), 69 `method`), 95  
`forward()` (`mwptoolkit.module.Attention.multi_head_attention.MultiHeadAttention` `forward()` (`mwptoolkit.module.Embedder.position_embedder.EPTPosition`  
`method`), 70 `method`), 96  
`forward()` (`mwptoolkit.module.Attention.multi_head_attention.MultiHeadAttentionWeights` `forward()` (`mwptoolkit.module.Embedder.position_embedder.PositionalEn`  
`method`), 71 `method`), 97  
`forward()` (`mwptoolkit.module.Attention.multi_head_attention.MultiHeadAttention` `forward()` (`mwptoolkit.module.Embedder.position_embedder.PositionEmb`  
`method`), 72 `method`), 97  
`forward()` (`mwptoolkit.module.Attention.self_attention.SelfAttention` `forward()` (`mwptoolkit.module.Embedder.position_embedder.PositionEmb`  
`method`), 72 `method`), 97  
`forward()` (`mwptoolkit.module.Attention.self_attention.SelfAttentionMask` `forward()` (`mwptoolkit.module.Embedder.roberta_embedder.RobertaEmbe`  
`method`), 73 `method`), 98  
`forward()` (`mwptoolkit.module.Attention.seq_attention.Attention` `forward()` (`mwptoolkit.module.Encoder.graph_based_encoder.GraphBase`  
`method`), 73 `method`), 98  
`forward()` (`mwptoolkit.module.Attention.seq_attention.MaskedAttention` `forward()` (`mwptoolkit.module.Encoder.graph_based_encoder.GraphBase`  
`method`), 74 `method`), 99  
`forward()` (`mwptoolkit.module.Attention.seq_attention.RelaxedAttention` `forward()` (`mwptoolkit.module.Encoder.graph_based_encoder.GraphEnco`  
`method`), 74 `method`), 99  
`forward()` (`mwptoolkit.module.Attention.seq_attention.SequenceAttention` `forward()` (`mwptoolkit.module.Encoder.graph_based_encoder.NumEncod`  
`method`), 75 `method`), 99  
`forward()` (`mwptoolkit.module.Attention.tree_attention.TreeAttention` `forward()` (`mwptoolkit.module.Encoder.rnn_encoder.BasicRNNEncoder`  
`method`), 75 `method`), 100  
`forward()` (`mwptoolkit.module.Decoder.ept_decoder.AveragePooling` `forward()` (`mwptoolkit.module.Encoder.rnn_encoder.GroupAttentionRNN`  
`method`), 76 `method`), 101  
`forward()` (`mwptoolkit.module.Decoder.ept_decoder.Decoder` `forward()` (`mwptoolkit.module.Encoder.rnn_encoder.HWCPDecoder`  
`method`), 76 `method`), 101  
`forward()` (`mwptoolkit.module.Decoder.ept_decoder.LogSoftmax` `forward()` (`mwptoolkit.module.Encoder.rnn_encoder.SalignedEncoder`  
`method`), 83 `method`), 102  
`forward()` (`mwptoolkit.module.Decoder.ept_decoder.Squeeze` `forward()` (`mwptoolkit.module.Encoder.rnn_encoder.SelfAttentionRNNEn`  
`method`), 85 `method`), 102  
`forward()` (`mwptoolkit.module.Decoder.rnn_decoder.AttentionRNNDecoder` `forward()` (`mwptoolkit.module.Encoder.transformer_encoder.BertEncoder`  
`method`), 88 `method`), 103  
`forward()` (`mwptoolkit.module.Decoder.rnn_decoder.BasicRNNDecoder` `forward()` (`mwptoolkit.module.Encoder.transformer_encoder.GroupATTE`  
`method`), 89 `method`), 103  
`forward()` (`mwptoolkit.module.Decoder.rnn_decoder.SalignedRNNDecoder` `forward()` (`mwptoolkit.module.Encoder.transformer_encoder.Transformer`  
`method`), 89 `method`), 104  
`forward()` (`mwptoolkit.module.Decoder.transformer_decoder.TransformerDecoder` `forward()` (`mwptoolkit.module.Graph.gcn.GCN`  
`method`), 90 `method`), 106  
`forward()` (`mwptoolkit.module.Decoder.tree_decoder.HMSolver` `forward()` (`mwptoolkit.module.Graph.graph_module.Graph_Module`  
`method`), 91 `method`), 106  
`forward()` (`mwptoolkit.module.Decoder.tree_decoder.LSTMTreeDecoder` `forward()` (`mwptoolkit.module.Graph.graph_module.Num_Graph_Module`  
`method`), 92 `method`), 107

`forward()` (`mwptoolkit.module.Graph.graph_module.ParseGraphModule` method), 107  
`forward()` (`mwptoolkit.module.Layer.graph_layers.GraphConvolution` method), 108  
`forward()` (`mwptoolkit.module.Layer.graph_layers.LayerNorm` method), 108  
`forward()` (`mwptoolkit.module.Layer.graph_layers.MeanAggregation` method), 108  
`forward()` (`mwptoolkit.module.Layer.graph_layers.PositionwiseFeedForward` method), 109  
`forward()` (`mwptoolkit.module.Layer.layers.GenVar` method), 109  
`forward()` (`mwptoolkit.module.Layer.layers.Transformer` method), 110  
`forward()` (`mwptoolkit.module.Layer.layers.TreeAttnDecoderRNN` method), 110  
`forward()` (`mwptoolkit.module.Layer.transformer_layer.EPTTransformerUtils` method), 111  
`forward()` (`mwptoolkit.module.Layer.transformer_layer.GATEWAY` method), 111  
`forward()` (`mwptoolkit.module.Layer.transformer_layer.LayerNorm` method), 111  
`forward()` (`mwptoolkit.module.Layer.transformer_layer.PositionwiseFeedForward` method), 112  
`forward()` (`mwptoolkit.module.Layer.transformer_layer.SublayerController` method), 112  
`forward()` (`mwptoolkit.module.Layer.transformer_layer.TransformInfixToPrefix` method), 113  
`forward()` (`mwptoolkit.module.Layer.tree_layers.Dec_LSTM` method), 113  
`forward()` (`mwptoolkit.module.Layer.tree_layers.DecomposeModel` method), 114  
`forward()` (`mwptoolkit.module.Layer.tree_layers.DQN` method), 113  
`forward()` (`mwptoolkit.module.Layer.tree_layers.GateNN` method), 114  
`forward()` (`mwptoolkit.module.Layer.tree_layers.GenerateNode` method), 114  
`forward()` (`mwptoolkit.module.Layer.tree_layers.Merge` method), 115  
`forward()` (`mwptoolkit.module.Layer.tree_layers.NodeEmbeddingLayer` method), 115  
`forward()` (`mwptoolkit.module.Layer.tree_layers.NodeGenerator` method), 116  
`forward()` (`mwptoolkit.module.Layer.tree_layers.Prediction` method), 116  
`forward()` (`mwptoolkit.module.Layer.tree_layers.RecursiveFun` method), 117  
`forward()` (`mwptoolkit.module.Layer.tree_layers.Score` method), 117  
`forward()` (`mwptoolkit.module.Layer.tree_layers.ScoreModel` method), 117  
`forward()` (`mwptoolkit.module.Layer.tree_layers.SemanticFun` method), 118

`forward()` (`mwptoolkit.module.Layer.tree_layers.SubTreeMerger` method), 118  
`forward()` (`mwptoolkit.module.Layer.tree_layers.TreeAttention` method), 118  
`forward()` (`mwptoolkit.module.Layer.tree_layers.TreeEmbeddingModel` method), 119  
`forward_beam()` (`mwptoolkit.module.Decoder.tree_decoder.HMSDecoder` method), 91  
`forward_step()` (`mwptoolkit.module.Decoder.tree_decoder.HMSDecoder` method), 91  
`forward_teacher()` (`mwptoolkit.module.Decoder.tree_decoder.HMSDecoder` method), 91  
`fraction_word_to_num()` (`in module mwptoolkit.utils.preprocess_tool.number_operator`), 151  
`FRACTIONAL_PATTERN` (`mwptoolkit.utils.enum_type.EPT` attribute), 143  
`from_infix_to_multi_way_tree()` (`in module mwptoolkit.utils.preprocess_tool.equation_operator`), 149  
`from_infix_to_postfix()` (`in module mwptoolkit.utils.preprocess_tool.equation_operator`), 149  
`from_infix_to_prefix()` (`in module mwptoolkit.utils.preprocess_tool.equation_operator`), 149  
`from_postfix_to_infix()` (`in module mwptoolkit.utils.preprocess_tool.equation_operator`), 149  
`from_postfix_to_prefix()` (`in module mwptoolkit.utils.preprocess_tool.equation_operator`), 149  
`from_prefix_to_infix()` (`in module mwptoolkit.utils.preprocess_tool.equation_operator`), 149  
`from_prefix_to_postfix()` (`in module mwptoolkit.utils.preprocess_tool.equation_operator`), 149  
`fully_supervised` (`mwptoolkit.utils.enum_type.SupervisingMode` attribute), 147  
`FUN_END_EQN` (`mwptoolkit.utils.enum_type.EPT` attribute), 143  
`FUN_END_EQN_ID` (`mwptoolkit.utils.enum_type.EPT` attribute), 143  
`FUN_EQ_SGN_ID` (`mwptoolkit.utils.enum_type.EPT` attribute), 143  
`FUN_NEW_EQN` (`mwptoolkit.utils.enum_type.EPT` attribute), 143  
`FUN_NEW_EQN_ID` (`mwptoolkit.utils.enum_type.EPT` attribute), 144



FUN\_NEW\_VAR (*mwptoolkit.utils.enum\_type.EPT attribute*), 144  
 FUN\_NEW\_VAR\_ID (*mwptoolkit.utils.enum\_type.EPT attribute*), 144  
 FUN\_TOKENS (*mwptoolkit.utils.enum\_type.EPT attribute*), 144  
 FUN\_TOKENS\_WITH\_EQ (*mwptoolkit.utils.enum\_type.EPT attribute*), 144  
 function\_arities (*mwptoolkit.module.Decoder.ept\_decoder.ExpressionDecoderModule attribute*), 79

## G

GAEncoderLayer (*class in mwptoolkit.module.Layer.transformer\_layer*), 111  
 GateNN (*class in mwptoolkit.module.Layer.tree\_layers*), 114  
 gather\_vectors() (*mwptoolkit.model.Seq2Seq.ept.EPT method*), 39  
 GCN (*class in mwptoolkit.module.Graph.gcn*), 106  
 gelu() (*mwptoolkit.module.Layer.transformer\_layer.TransformerLayer method*), 113  
 generate() (*mwptoolkit.module.Strategy.beam\_search.Beam\_Search\_Hypothesis method*), 120  
 generate\_decoder\_input() (*mwptoolkit.model.Graph2Tree.multiencdec.MultiEncDec method*), 63  
 generate\_tree\_input() (*mwptoolkit.model.Graph2Tree.graph2tree.Graph2Tree method*), 61  
 generate\_tree\_input() (*mwptoolkit.model.Graph2Tree.multiencdec.MultiEncDec method*), 63  
 generate\_tree\_input() (*mwptoolkit.model.Seq2Tree.berttd.BertTD method*), 49  
 generate\_tree\_input() (*mwptoolkit.model.Seq2Tree.gts.GTS method*), 51  
 generate\_tree\_input() (*mwptoolkit.model.Seq2Tree.mwpbert.MWPBert method*), 52  
 generate\_tree\_input() (*mwptoolkit.model.Seq2Tree.sausolver.SAUSolver method*), 54  
 generate\_tree\_input() (*mwptoolkit.model.Seq2Tree.treelstm.TreeLSTM method*), 55  
 generate\_tree\_input() (*mwptoolkit.model.Seq2Tree.tsn.TSN method*), 58  
 GenerateNode (*class in mwptoolkit.module.Layer.tree\_layers*), 114  
 GenVar (*class in mwptoolkit.module.Layer.layers*), 109  
 get\_adj() (*mwptoolkit.module.Graph.graph\_module.Graph\_Module method*), 107  
 get\_all\_number\_encoder\_outputs() (*mwptoolkit.model.Graph2Tree.graph2tree.Graph2Tree method*), 62  
 get\_all\_number\_encoder\_outputs() (*mwptoolkit.model.Graph2Tree.multiencdec.MultiEncDec method*), 63  
 get\_all\_number\_encoder\_outputs() (*mwptoolkit.model.Seq2Tree.berttd.BertTD method*), 49  
 get\_all\_number\_encoder\_outputs() (*mwptoolkit.model.Seq2Tree.gts.GTS method*), 51  
 get\_all\_number\_encoder\_outputs() (*mwptoolkit.model.Seq2Tree.mwpbert.MWPBert method*), 52  
 get\_all\_number\_encoder\_outputs() (*mwptoolkit.model.Seq2Tree.sausolver.SAUSolver method*), 54  
 get\_all\_number\_encoder\_outputs() (*mwptoolkit.model.Seq2Tree.treelstm.TreeLSTM method*), 55  
 get\_all\_number\_encoder\_outputs() (*mwptoolkit.model.Seq2Tree.tsn.TSN method*), 58  
 get\_class\_embedding\_mask() (*mwptoolkit.module.Decoder.tree\_decoder.HMSDecoder method*), 91  
 get\_data\_loader\_module() (*in module mwptoolkit.data.utils*), 22  
 get\_dataset\_module() (*in module mwptoolkit.data.utils*), 22  
 get\_deprel\_tree() (*in module mwptoolkit.utils.preprocess\_tool.sentence\_operator*), 154  
 get\_deprel\_tree\_() (*in module mwptoolkit.utils.preprocess\_tool.sentence\_operator*), 155  
 get\_embedding() (*mwptoolkit.module.Embedder.position\_embedder.PositionEmbedder method*), 97  
 get\_embedding\_without\_pad() (*in module mwptoolkit.module.Decoder.ept\_decoder*), 87  
 get\_evaluator() (*in module mwptoolkit.evaluate.evaluator*), 30  
 get\_evaluator\_module() (*in module mwptoolkit.evaluate.evaluator*), 30  
 get\_fix\_constant() (*mwptoolkit.module.Encoder.rnn\_encoder.SalignedEncoder method*), 102



GPT2 (class in mwptoolkit.model.PreTrain.gpt2), 65  
 Graph2Tree (class in mwptoolkit.model.Graph2Tree.graph2tree), 61  
 Graph2TreeTrainer (class in mwptoolkit.trainer.supervised\_trainer), 127  
 Graph\_Module (class in mwptoolkit.module.Graph.graph\_module), 106  
 GraphBasedEncoder (class in mwptoolkit.module.Encoder.graph\_based\_encoder), 98  
 GraphBasedMultiEncoder (class in mwptoolkit.module.Encoder.graph\_based\_encoder), 99  
 GraphConvolution (class in mwptoolkit.module.Layer.graph\_layers), 108  
 GraphEncoder (class in mwptoolkit.module.Encoder.graph\_based\_encoder), 99  
 greedy\_search() (in module mwptoolkit.module.Strategy.greedy), 121  
 group\_mask() (in module mwptoolkit.module.Attention.group\_attention), 70  
 GroupATT (class in mwptoolkit.model.Seq2Seq.groupatt), 41  
 GroupATTencoder (class in mwptoolkit.module.Encoder.transformer\_encoder), 103  
 GroupAttention (class in mwptoolkit.module.Attention.group\_attention), 69  
 GroupAttentionRNNEncoder (class in mwptoolkit.module.Encoder.rnn\_encoder), 100  
 GTS (class in mwptoolkit.model.Seq2Tree.gts), 50  
 GTSTrainer (class in mwptoolkit.trainer.supervised\_trainer), 126  
**H**  
 hidden\_dim (mwptoolkit.module.Attention.multi\_head\_attention.EPTMultiHeadAttentionWeights property), 72  
 HMSDecoder (class in mwptoolkit.module.Decoder.tree\_decoder), 91  
 HMSTrainer (class in mwptoolkit.trainer.supervised\_trainer), 128  
 hmwp (mwptoolkit.utils.enum\_type.DatasetName attribute), 142  
 HWCPEncoder (class in mwptoolkit.module.Encoder.rnn\_encoder), 101  
 hyper\_search\_process() (in module mwptoolkit.hyper\_search), 157  
**I**  
 id\_reedit() (in module mwptoolkit.utils.preprocess\_tool.dataset\_operator), 148  
 IN\_EQN (mwptoolkit.utils.enum\_type.EPT attribute), 144  
 IN\_TNPAD (mwptoolkit.utils.enum\_type.EPT attribute), 144  
 IN\_TNUM (mwptoolkit.utils.enum\_type.EPT attribute), 144  
 IN\_TPAD (mwptoolkit.utils.enum\_type.EPT attribute), 144  
 IN\_TXT (mwptoolkit.utils.enum\_type.EPT attribute), 144  
 Infix (mwptoolkit.utils.enum\_type.FixType attribute), 146  
 infix\_to\_postfix() (in module mwptoolkit.utils.preprocess\_tool.equation\_operator), 150  
 InfixEvaluator (class in mwptoolkit.evaluate.evaluator), 25  
 init\_batches() (mwptoolkit.data.dataloader.abstract\_dataloader.AbstractDataLoader method), 4  
 init\_batches() (mwptoolkit.data.dataloader.multi\_equation\_dataloader.MultiEquation method), 7  
 init\_batches() (mwptoolkit.data.dataloader.pretrain\_dataloader.PretrainDataLoader method), 7  
 init\_batches() (mwptoolkit.data.dataloader.single\_equation\_dataloader.SingleEquation method), 9  
 init\_batches() (mwptoolkit.data.dataloader.template\_dataloader.TemplateDataLoader method), 9  
 init\_decoder\_inputs() (mwptoolkit.model.Seq2Seq.dns.DNS method), 38  
 init\_decoder\_inputs() (mwptoolkit.model.Seq2Seq.groupatt.GroupATT method), 41  
 init\_decoder\_inputs() (mwptoolkit.model.Seq2Seq.rnnencdec.RNNEncDec method), 44  
 init\_decoder\_inputs() (mwptoolkit.model.Seq2Seq.rnnvae.RNNVAE method), 45  
 init\_decoder\_inputs() (mwptoolkit.model.Seq2Seq.transformer.Transformer method), 48  
 init\_embedding\_params() (mwptoolkit.module.Embedder.basic\_embedder.BasicEmbedder method), 95  
 init\_encoder\_mask() (mwptoolkit.model.Seq2Tree.tsn.TSN method), 148

58  
 init\_factor() (mwp- toolkit.module.Layer.graph\_layers), 108  
 toolkit.module.Decoder.ept\_decoder.DecoderModel (class in mwp-  
 toolkit.module.Layer.transformer\_layer),  
 method), 77 111  
 init\_hidden() (mwp- lca() (mwptoolkit.utils.data\_structure.GoldTree  
 toolkit.module.Decoder.rnn\_decoder.AttentionalRNNDerivator method), 141  
 method), 88 leaf\_emb() (mwptoolkit.module.Layer.tree\_layers.RecursiveNN  
 method), 117  
 init\_hidden() (mwp- lists2dict() (in module mwptoolkit.utils.utils), 156  
 toolkit.module.Decoder.rnn\_decoder.BasicRNNEncoder method), 66  
 method), 89 load\_data() (mwptoolkit.data.dataloader.abstract\_dataloader.AbstractData  
 method), 4  
 init\_hidden() (mwp- load\_data() (mwptoolkit.data.dataloader.multi\_equation\_dataloader.MultiEquationData  
 toolkit.module.Encoder.rnn\_encoder.SelfAttentionRNNEncoder method), 7  
 method), 103 load\_data() (mwptoolkit.data.dataloader.pretrain\_dataloader.PretrainData  
 method), 8  
 init\_logger() (in module mwptoolkit.utils.logger), 148  
 init\_seed() (in module mwptoolkit.utils.utils), 156  
 init\_seq2seq\_decoder\_inputs() (mwp- load\_data() (mwptoolkit.data.dataloader.single\_equation\_dataloader.SingleEquationData  
 toolkit.model.Seq2Tree.trnn.TRNN method), 9  
 method), 56 load\_data() (mwptoolkit.data.dataloader.template\_dataloader.TemplateData  
 method), 10  
 init\_soft\_target() (mwp- load\_from\_pretrained() (mwp-  
 toolkit.model.Seq2Tree.tsn.TSN method), toolkit.config.configuration.Config class  
 58 method), 2  
 init\_stacks() (mwp- load\_from\_pretrained() (mwp-  
 toolkit.module.Decoder.tree\_decoder.HMSDecoder toolkit.data.dataset.abstract\_dataset.AbstractDataset  
 method), 92 class method), 11  
 initialize\_fix\_constant() (mwp- load\_from\_pretrained() (mwp-  
 toolkit.module.Encoder.rnn\_encoder.SalignedEncoder toolkit.data.dataset.dataset\_ept.DatasetEPT  
 method), 102 class method), 13  
 is\_equal() (mwptoolkit.utils.data\_structure.GoldTree load\_from\_pretrained() (mwp-  
 method), 141 toolkit.data.dataset.dataset\_hms.DatasetHMS  
 class method), 14  
 is\_expression\_type (mwp- load\_from\_pretrained() (mwp-  
 toolkit.module.Decoder.ept\_decoder.DecoderModel method), 15  
 property), 77 toolkit.data.dataset.dataset\_multiencdec.DatasetMultiEncDec  
 class method), 15  
 is\_float() (mwptoolkit.utils.data\_structure.GoldTree load\_from\_pretrained() (mwp-  
 method), 141 toolkit.data.dataset.multi\_equation\_dataset.MultiEquationDataset  
 class method), 17  
 is\_in\_rel\_quants() (mwp- load\_from\_pretrained() (mwp-  
 toolkit.utils.data\_structure.GoldTree method), toolkit.data.dataset.pretrain\_dataset.PretrainDataset  
 141 class method), 18  
**J**  
 joint\_fraction() (in module mwp- load\_from\_pretrained() (mwp-  
 toolkit.utils.preprocess\_tool.number\_operator), toolkit.data.dataset.single\_equation\_dataset.SingleEquationDataset  
 151 class method), 20  
 joint\_number() (in module mwp- load\_from\_pretrained() (mwp-  
 toolkit.utils.preprocess\_tool.number\_operator), toolkit.data.dataset.template\_dataset.TemplateDataset  
 152 class method), 22  
 joint\_number\_() (in module mwp- load\_next\_batch() (mwp-  
 toolkit.utils.preprocess\_tool.number\_operator), toolkit.data.dataloader.abstract\_dataloader.AbstractDataLoader  
 152 method), 4  
**L**  
 load\_next\_batch() (mwp-  
 toolkit.data.dataloader.multi\_equation\_dataloader.MultiEquationDataLoader  
 method), 7  
 LayerNorm (class in mwp-



`load_next_batch()` (`mwptoolkit.data.dataloader.pretrain_dataloader.PretrainDataLoader` method), 8  
`load_next_batch()` (`mwptoolkit.data.dataloader.single_equation_dataloader.SingleEquationDataLoader` method), 9  
`load_next_batch()` (`mwptoolkit.data.dataloader.template_dataloader.TemplateDataLoader` method), 10  
`LogSoftmax` (class in `mwptoolkit.module.Decoder.ept_decoder`), 83  
`LSTMBasedTreeDecoder` (class in `mwptoolkit.module.Decoder.tree_decoder`), 92

## M

`mask2num()` (`mwptoolkit.model.Seq2Tree.trnn.TRNN` method), 56  
`mask_forward()` (in module `mwptoolkit.module.Decoder.ept_decoder`), 87  
`masked_cross_entropy()` (in module `mwptoolkit.loss.masked_cross_entropy_loss`), 32  
`MaskedCrossEntropyLoss` (class in `mwptoolkit.loss.masked_cross_entropy_loss`), 32  
`MaskedRelevantScore` (class in `mwptoolkit.module.Attention.seq_attention`), 74  
`MaskSymbol` (class in `mwptoolkit.utils.enum_type`), 146  
`math23k` (`mwptoolkit.utils.enum_type.DatasetName` attribute), 142  
`MathEN` (class in `mwptoolkit.model.Seq2Seq.mathen`), 42  
`mawps` (`mwptoolkit.utils.enum_type.DatasetName` attribute), 142  
`mawps_asdiv_a_svamp` (`mwptoolkit.utils.enum_type.DatasetName` attribute), 142  
`mawps_single` (`mwptoolkit.utils.enum_type.DatasetName` attribute), 143  
`MeanAggregator` (class in `mwptoolkit.module.Layer.graph_layers`), 108  
`MEM_MAX` (`mwptoolkit.utils.enum_type.EPT` attribute), 144  
`MEM_PREFIX` (`mwptoolkit.utils.enum_type.EPT` attribute), 144  
`Merge` (class in `mwptoolkit.module.Layer.tree_layers`), 115  
`merge()` (`mwptoolkit.module.Layer.tree_layers.TreeEmbeddingModule` method), 119  
`MODEL_EXPR_PTR_TRANS` (`mwptoolkit.utils.enum_type.EPT` attribute), 144  
`MODEL_EXPR_TRANS` (`mwptoolkit.utils.enum_type.EPT` attribute), 144  
`model_test()` (`mwptoolkit.model.Graph2Tree.graph2tree.Graph2Tree` method), 62  
`model_test()` (`mwptoolkit.model.Graph2Tree.multienccdec.MultiEncDec` method), 63  
`model_test()` (`mwptoolkit.model.PreTrain.bertgen.BERTGen` method), 64  
`model_test()` (`mwptoolkit.model.PreTrain.gpt2.GPT2` method), 66  
`model_test()` (`mwptoolkit.model.PreTrain.robertagen.RobertaGen` method), 67  
`model_test()` (`mwptoolkit.model.Seq2Seq.dns.DNS` method), 38  
`model_test()` (`mwptoolkit.model.Seq2Seq.ept.EPT` method), 40  
`model_test()` (`mwptoolkit.model.Seq2Seq.groupatt.GroupATT` method), 41  
`model_test()` (`mwptoolkit.model.Seq2Seq.mathen.MathEN` method), 43  
`model_test()` (`mwptoolkit.model.Seq2Seq.rnnencdec.RNNEncDec` method), 44  
`model_test()` (`mwptoolkit.model.Seq2Seq.rnnvae.RNNVAE` method), 45  
`model_test()` (`mwptoolkit.model.Seq2Seq.saligned.Saligned` method), 47  
`model_test()` (`mwptoolkit.model.Seq2Seq.transformer.Transformer` method), 48  
`model_test()` (`mwptoolkit.model.Seq2Tree.berttd.BertTD` method), 49  
`model_test()` (`mwptoolkit.model.Seq2Tree.gts.GTS` method), 51  
`model_test()` (`mwptoolkit.model.Seq2Tree.mwpbert.MWPBert` method), 52  
`model_test()` (`mwptoolkit.model.Seq2Tree.sausolver.SAUSolver` method), 54  
`model_test()` (`mwptoolkit.model.Seq2Tree.treelstm.TreeLSTM` method), 55  
`model_test()` (`mwptoolkit.model.Seq2Tree.trnn.TRNN` method), 56  
`model_test()` (`mwptoolkit.model.Seq2Tree.tsn.TSN` method), 58  
`MODEL_VANILLA_TRANS` (`mwptoolkit.utils.enum_type.EPT` attribute), 144  
`module`  
`mwptoolkit.config.configuration`, 1  
`mwptoolkit.data.dataloader.abstract_dataloader`, 3  
`mwptoolkit.data.dataloader.dataloader_ept`, 4  
`mwptoolkit.data.dataloader.dataloader_hms`, 5  
`mwptoolkit.data.dataloader.dataloader_multiencdec`, 5  
`mwptoolkit.data.dataloader.multi_equation_dataloader`, 6  
`mwptoolkit.data.dataloader.pretrain_dataloader`, 7

<code>mwptoolkit.data.dataloader.single_equation_dataset</code> , 8	<code>mwptoolkit.module.Attention.group_attention</code> , 69
<code>mwptoolkit.data.dataloader.template_dataloader</code> , 9	<code>mwptoolkit.module.Attention.multi_head_attention</code> , 70
<code>mwptoolkit.data.dataset.abstract_dataset</code> , 10	<code>mwptoolkit.module.Attention.self_attention</code> , 72
<code>mwptoolkit.data.dataset.dataset_ept</code> , 12	<code>mwptoolkit.module.Attention.seq_attention</code> , 73
<code>mwptoolkit.data.dataset.dataset_hms</code> , 13	<code>mwptoolkit.module.Attention.tree_attention</code> , 75
<code>mwptoolkit.data.dataset.dataset_multienodec</code> , 14	<code>mwptoolkit.module.Decoder.ept_decoder</code> , 75
<code>mwptoolkit.data.dataset.multi_equation_dataset</code> , 16	<code>mwptoolkit.module.Decoder.rnn_decoder</code> , 88
<code>mwptoolkit.data.dataset.pretrain_dataset</code> , 17	<code>mwptoolkit.module.Decoder.transformer_decoder</code> , 90
<code>mwptoolkit.data.dataset.single_equation_dataset</code> , 19	<code>mwptoolkit.module.Decoder.tree_decoder</code> , 91
<code>mwptoolkit.data.dataset.template_dataset</code> , 20	<code>mwptoolkit.module.Embedder.basic_embedder</code> , 95
<code>mwptoolkit.data.utils</code> , 22	<code>mwptoolkit.module.Embedder.bert_embedder</code> , 95
<code>mwptoolkit.evaluate.evaluator</code> , 25	<code>mwptoolkit.module.Embedder.position_embedder</code> , 95
<code>mwptoolkit.hyper_search</code> , 157	<code>mwptoolkit.module.Embedder.roberta_embedder</code> , 98
<code>mwptoolkit.loss.abstract_loss</code> , 31	<code>mwptoolkit.module.Encoder.graph_based_encoder</code> , 98
<code>mwptoolkit.loss.binary_cross_entropy_loss</code> , 31	<code>mwptoolkit.module.Encoder.rnn_encoder</code> , 100
<code>mwptoolkit.loss.cross_entropy_loss</code> , 32	<code>mwptoolkit.module.Encoder.transformer_encoder</code> , 103
<code>mwptoolkit.loss.masked_cross_entropy_loss</code> , 32	<code>mwptoolkit.module.Environment.stack_machine</code> , 104
<code>mwptoolkit.loss.mse_loss</code> , 33	<code>mwptoolkit.module.Graph.gcn</code> , 106
<code>mwptoolkit.loss.nll_loss</code> , 33	<code>mwptoolkit.module.Graph.graph_module</code> , 106
<code>mwptoolkit.loss.smoothed_cross_entropy_loss</code> , 34	<code>mwptoolkit.module.Layer.graph_layers</code> , 108
<code>mwptoolkit.model.Graph2Tree.graph2tree</code> , 61	<code>mwptoolkit.module.Layer.layers</code> , 109
<code>mwptoolkit.model.Graph2Tree.multienodec</code> , 62	<code>mwptoolkit.module.Layer.transformer_layer</code> , 110
<code>mwptoolkit.model.PreTrain.bertgen</code> , 64	<code>mwptoolkit.module.Layer.tree_layers</code> , 113
<code>mwptoolkit.model.PreTrain.gpt2</code> , 65	<code>mwptoolkit.module.Strategy.beam_search</code> , 119
<code>mwptoolkit.model.PreTrain.robertagen</code> , 66	<code>mwptoolkit.module.Strategy.greedy</code> , 121
<code>mwptoolkit.model.Seq2Seq.dns</code> , 37	<code>mwptoolkit.module.Strategy.sampling</code> , 121
<code>mwptoolkit.model.Seq2Seq.ept</code> , 39	<code>mwptoolkit.quick_start</code> , 159
<code>mwptoolkit.model.Seq2Seq.groupatt</code> , 41	<code>mwptoolkit.trainer.abstract_trainer</code> , 123
<code>mwptoolkit.model.Seq2Seq.mathen</code> , 42	<code>mwptoolkit.trainer.supervised_trainer</code> , 124
<code>mwptoolkit.model.Seq2Seq.rnnencdec</code> , 43	<code>mwptoolkit.trainer.template_trainer</code> , 139
<code>mwptoolkit.model.Seq2Seq.rnnvae</code> , 45	<code>mwptoolkit.utils.data_structure</code> , 141
<code>mwptoolkit.model.Seq2Seq.saligned</code> , 46	<code>mwptoolkit.utils.enum_type</code> , 142
<code>mwptoolkit.model.Seq2Seq.transformer</code> , 47	<code>mwptoolkit.utils.logger</code> , 148
<code>mwptoolkit.model.Seq2Tree.berttd</code> , 49	<code>mwptoolkit.utils.preprocess_tool.dataset_operator</code> , 148
<code>mwptoolkit.model.Seq2Tree.gts</code> , 50	<code>mwptoolkit.utils.preprocess_tool.equation_operator</code> , 148
<code>mwptoolkit.model.Seq2Tree.mwpbert</code> , 51	
<code>mwptoolkit.model.Seq2Tree.sausolver</code> , 53	
<code>mwptoolkit.model.Seq2Tree.treelstm</code> , 54	
<code>mwptoolkit.model.Seq2Tree.trnn</code> , 56	
<code>mwptoolkit.model.Seq2Tree.tsn</code> , 57	

148 `mwptoolkit.data.dataloader.single_equation_dataloader`  
`mwptoolkit.utils.preprocess_tool.number_operator`, 8  
 151 `mwptoolkit.data.dataloader.template_dataloader`  
`mwptoolkit.utils.preprocess_tool.number_transformation`, 9  
 152 `mwptoolkit.data.dataset.abstract_dataset`  
`mwptoolkit.utils.preprocess_tool.sentence_operator`, 10  
 154 `mwptoolkit.data.dataset.dataset_ept`  
`mwptoolkit.utils.utils`, 155  
`mse_loss()` (`mwptoolkit.model.Seq2Tree.sausolver.SAUSolver`  
 method), 54  
`MSELoss` (class in `mwptoolkit.loss.mse_loss`), 33  
`Multi` (`mwptoolkit.utils.enum_type.Operators` attribute),  
 147  
`MultiEncDec` (class in `mwptoolkit.model.Graph2Tree.mutiencdec`), 62  
`MultiEncDecEvaluator` (class in `mwptoolkit.evaluate.evaluator`), 26  
`MultiEncDecTrainer` (class in `mwptoolkit.trainer.supervised_trainer`), 129  
`MultiEquation` (`mwptoolkit.utils.enum_type.TaskType`  
 attribute), 147  
`MultiEquationDataLoader` (class in `mwptoolkit.data.dataloader.multi_equation_dataloader`),  
 6  
`MultiEquationDataset` (class in `mwptoolkit.data.dataset.multi_equation_dataset`),  
 16  
`MultiHeadAttention` (class in `mwptoolkit.module.attention.multi_head_attention`),  
 72  
`MULTIPLES` (`mwptoolkit.utils.enum_type.EPT` attribute),  
 144  
`MultiWayTree` (`mwptoolkit.utils.enum_type.FixType` at-  
 tribute), 146  
`MultiWayTreeEvaluator` (class in `mwptoolkit.evaluate.evaluator`), 28  
`MWPBert` (class in `mwptoolkit.model.Seq2Tree.mwpbert`),  
 51  
`MWPBertTrainer` (class in `mwptoolkit.trainer.supervised_trainer`), 128  
`mwptoolkit.config.configuration`  
 module, 1  
`mwptoolkit.data.dataloader.abstract_dataloader`  
 module, 3  
`mwptoolkit.data.dataloader.dataloader_ept`  
 module, 4  
`mwptoolkit.data.dataloader.dataloader_hms`  
 module, 5  
`mwptoolkit.data.dataloader.dataloader_multiencdec`  
 module, 5  
`mwptoolkit.data.dataloader.multi_equation_dataloader`  
 module, 6  
`mwptoolkit.data.dataloader.pretrain_dataloader`  
 module, 7  
`mwptoolkit.data.dataloader.single_equation_dataloader`  
 module, 8  
`mwptoolkit.data.dataloader.template_dataloader`  
 module, 9  
`mwptoolkit.data.dataset.abstract_dataset`  
 module, 10  
`mwptoolkit.data.dataset.dataset_ept`  
 module, 12  
`mwptoolkit.data.dataset.dataset_hms`  
 module, 13  
`mwptoolkit.data.dataset.dataset_multiencdec`  
 module, 14  
`mwptoolkit.data.dataset.multi_equation_dataset`  
 module, 16  
`mwptoolkit.data.dataset.pretrain_dataset`  
 module, 17  
`mwptoolkit.data.dataset.single_equation_dataset`  
 module, 19  
`mwptoolkit.data.dataset.template_dataset`  
 module, 20  
`mwptoolkit.data.utils`  
 module, 22  
`mwptoolkit.evaluate.evaluator`  
 module, 25  
`mwptoolkit.hyper_search`  
 module, 157  
`mwptoolkit.loss.abstract_loss`  
 module, 31  
`mwptoolkit.loss.binary_cross_entropy_loss`  
 module, 31  
`mwptoolkit.loss.cross_entropy_loss`  
 module, 32  
`mwptoolkit.loss.masked_cross_entropy_loss`  
 module, 32  
`mwptoolkit.loss.mse_loss`  
 module, 33  
`mwptoolkit.loss.nll_loss`  
 module, 33  
`mwptoolkit.loss.smoothed_cross_entropy_loss`  
 module, 34  
`mwptoolkit.model.Graph2Tree.graph2tree`  
 module, 61  
`mwptoolkit.model.Graph2Tree.mutiencdec`  
 module, 62  
`mwptoolkit.model.PreTrain.bertgen`  
 module, 64  
`mwptoolkit.model.PreTrain.gpt2`  
 module, 65  
`mwptoolkit.model.PreTrain.robertagen`  
 module, 66  
`mwptoolkit.model.Seq2Seq.dns`  
 module, 37  
`mwptoolkit.model.Seq2Seq.ept`  
 module, 39

<code>mwptoolkit.model.Seq2Seq.groupatt</code> module, 41	<code>mwptoolkit.module.Encoder.rnn_encoder</code> module, 100
<code>mwptoolkit.model.Seq2Seq.mathen</code> module, 42	<code>mwptoolkit.module.Encoder.transformer_encoder</code> module, 103
<code>mwptoolkit.model.Seq2Seq.rnnencdec</code> module, 43	<code>mwptoolkit.module.Environment.stack_machine</code> module, 104
<code>mwptoolkit.model.Seq2Seq.rnnvae</code> module, 45	<code>mwptoolkit.module.Graph.gcn</code> module, 106
<code>mwptoolkit.model.Seq2Seq.saligned</code> module, 46	<code>mwptoolkit.module.Graph.graph_module</code> module, 106
<code>mwptoolkit.model.Seq2Seq.transformer</code> module, 47	<code>mwptoolkit.module.Layer.graph_layers</code> module, 108
<code>mwptoolkit.model.Seq2Tree.berttd</code> module, 49	<code>mwptoolkit.module.Layer.layers</code> module, 109
<code>mwptoolkit.model.Seq2Tree.gts</code> module, 50	<code>mwptoolkit.module.Layer.transformer_layer</code> module, 110
<code>mwptoolkit.model.Seq2Tree.mwpbert</code> module, 51	<code>mwptoolkit.module.Layer.tree_layers</code> module, 113
<code>mwptoolkit.model.Seq2Tree.sausolver</code> module, 53	<code>mwptoolkit.module.Strategy.beam_search</code> module, 119
<code>mwptoolkit.model.Seq2Tree.treelstm</code> module, 54	<code>mwptoolkit.module.Strategy.greedy</code> module, 121
<code>mwptoolkit.model.Seq2Tree.trnn</code> module, 56	<code>mwptoolkit.module.Strategy.sampling</code> module, 121
<code>mwptoolkit.model.Seq2Tree.tsn</code> module, 57	<code>mwptoolkit.quick_start</code> module, 159
<code>mwptoolkit.module.Attention.group_attention</code> module, 69	<code>mwptoolkit.trainer.abstract_trainer</code> module, 123
<code>mwptoolkit.module.Attention.multi_head_attention</code> module, 70	<code>mwptoolkit.trainer.supervised_trainer</code> module, 124
<code>mwptoolkit.module.Attention.self_attention</code> module, 72	<code>mwptoolkit.trainer.template_trainer</code> module, 139
<code>mwptoolkit.module.Attention.seq_attention</code> module, 73	<code>mwptoolkit.utils.data_structure</code> module, 141
<code>mwptoolkit.module.Attention.tree_attention</code> module, 75	<code>mwptoolkit.utils.enum_type</code> module, 142
<code>mwptoolkit.module.Decoder.ept_decoder</code> module, 75	<code>mwptoolkit.utils.logger</code> module, 148
<code>mwptoolkit.module.Decoder.rnn_decoder</code> module, 88	<code>mwptoolkit.utils.preprocess_tool.dataset_operator</code> module, 148
<code>mwptoolkit.module.Decoder.transformer_decoder</code> module, 90	<code>mwptoolkit.utils.preprocess_tool.equation_operator</code> module, 148
<code>mwptoolkit.module.Decoder.tree_decoder</code> module, 91	<code>mwptoolkit.utils.preprocess_tool.number_operator</code> module, 151
<code>mwptoolkit.module.Embedder.basic_embedder</code> module, 95	<code>mwptoolkit.utils.preprocess_tool.number_transfer</code> module, 152
<code>mwptoolkit.module.Embedder.bert_embedder</code> module, 95	<code>mwptoolkit.utils.preprocess_tool.sentence_operator</code> module, 154
<code>mwptoolkit.module.Embedder.position_embedder</code> module, 95	<code>mwptoolkit.utils.utils</code> module, 155
<code>mwptoolkit.module.Embedder.roberta_embedder</code> module, 98	
<code>mwptoolkit.module.Encoder.graph_based_encoder</code> module, 98	

## N

`NEG_INF` (*mwptoolkit.utils.enum\_type.EPT* attribute),  
144

NLLLoss (class in mwptoolkit.loss.nll\_loss), 33  
 Node (class in mwptoolkit.module.Layer.tree\_layers), 115  
 Node (class in mwptoolkit.utils.data\_structure), 142  
 NodeEmbeddingLayer (class in mwptoolkit.module.Layer.tree\_layers), 115  
 NodeEmbeddingNode (class in mwptoolkit.module.Layer.tree\_layers), 116  
 NodeGenerater (class in mwptoolkit.module.Layer.tree\_layers), 116  
 NON\_TOKEN (mwptoolkit.utils.enum\_type.SpecialTokens attribute), 147  
 Nonfix (mwptoolkit.utils.enum\_type.FixType attribute), 146  
 normalize() (mwptoolkit.module.Graph.graph\_module.GraphModule method), 107  
 normalize() (mwptoolkit.module.Graph.graph\_module.Num\_Graph\_Module method), 107  
 normalize() (mwptoolkit.module.Graph.graph\_module.Parse\_Graph\_Module method), 107  
 NUM (mwptoolkit.utils.enum\_type.MaskSymbol attribute), 146  
 NUM (mwptoolkit.utils.enum\_type.NumMask attribute), 146  
 Num\_Graph\_Module (class in mwptoolkit.module.Graph.graph\_module), 107  
 num\_heads (mwptoolkit.module.Attention.multi\_head\_attention module property), 72  
 NUM\_MAX (mwptoolkit.utils.enum\_type.EPT attribute), 144  
 num\_order\_processed() (mwptoolkit.data.dataloader.dataloader\_multiencdec.DataLoaderMultiEncDec method), 6  
 NUM\_PREFIX (mwptoolkit.utils.enum\_type.EPT attribute), 145  
 NUM\_TOKEN (mwptoolkit.utils.enum\_type.EPT attribute), 145  
 num\_transfer\_alg514() (in module mwptoolkit.utils.preprocess\_tool.number\_transfer), 152  
 num\_transfer\_draw() (in module mwptoolkit.utils.preprocess\_tool.number\_transfer), 152  
 num\_transfer\_hmwp() (in module mwptoolkit.utils.preprocess\_tool.number\_transfer), 153  
 num\_transfer\_multi() (in module mwptoolkit.utils.preprocess\_tool.number\_transfer), 153  
 number (mwptoolkit.utils.enum\_type.MaskSymbol attribute), 146  
 number (mwptoolkit.utils.enum\_type.NumMask attribute), 146  
 NUMBER\_AND\_FRACTION\_PATTERN (mwptoolkit.utils.enum\_type.EPT attribute), 144  
 NUMBER\_PATTERN (mwptoolkit.utils.enum\_type.EPT attribute), 144  
 NUMBER\_READINGS (mwptoolkit.utils.enum\_type.EPT attribute), 144  
 number\_transfer() (in module mwptoolkit.utils.preprocess\_tool.number\_transfer), 153  
 number\_transfer\_ape200k() (in module mwptoolkit.utils.preprocess\_tool.number\_transfer), 153  
 number\_transfer\_asdiv\_a() (in module mwptoolkit.utils.preprocess\_tool.number\_transfer), 153  
 number\_transfer\_math23k() (in module mwptoolkit.utils.preprocess\_tool.number\_transfer), 154  
 number\_transfer\_mawps() (in module mwptoolkit.utils.preprocess\_tool.number\_transfer), 154  
 number\_transfer\_mawps\_single() (in module mwptoolkit.utils.preprocess\_tool.number\_transfer), 154  
 number\_transfer\_single() (in module mwptoolkit.utils.preprocess\_tool.number\_transfer), 154  
 number\_transfer\_swap() (in module mwptoolkit.utils.preprocess\_tool.number\_transfer), 154  
 NumEncoder (class in mwptoolkit.module.Encoder.graph\_based\_encoder), 146  
 NumMask (class in mwptoolkit.utils.enum\_type), 146  
**O**  
 OpDecoderModel (class in mwptoolkit.module.Decoder.ept\_decoder), 83  
 operand\_norm (mwptoolkit.module.Decoder.ept\_decoder.ExpressionDecoder attribute), 79  
 operand\_out (mwptoolkit.module.Decoder.ept\_decoder.ExpressionPointer attribute), 81  
 operand\_out (mwptoolkit.module.Decoder.ept\_decoder.ExpressionTransformer attribute), 82  
 operand\_source\_embedding (mwptoolkit.module.Decoder.ept\_decoder.ExpressionDecoderModel attribute), 79  
 operand\_source\_factor (mwptoolkit.module.Decoder.ept\_decoder.ExpressionDecoderModel attribute), 79  
 operand\_word\_embedding (mwptoolkit.module.Decoder.ept\_decoder.ExpressionTransformer attribute), 82  
 OPERATIONS (class in mwptoolkit.module.Environment.stack\_machine), 104



`operator_mask()` (in module `mwptoolkit.utils.preprocess_tool.equation_operator`), 150  
`operator_mask_process()` (`mwptoolkit.data.dataset.abstract_dataset.AbstractDataset` method), 11  
`OPERATOR_PRECEDENCE` (`mwptoolkit.utils.enum_type.EPT` attribute), 145  
`Operators` (class in `mwptoolkit.utils.enum_type`), 147  
`OPERATORS` (`mwptoolkit.utils.enum_type.EPT` attribute), 145  
`OPT_TOKEN` (`mwptoolkit.utils.enum_type.SpecialTokens` attribute), 147  
`orig_infix_to_postfix()` (in module `mwptoolkit.utils.preprocess_tool.equation_operator`), 150  
`out_expression_expr()` (`mwptoolkit.model.Seq2Seq.ept.EPT` method), 40  
`out_expression_op()` (`mwptoolkit.model.Seq2Seq.ept.EPT` method), 40

**P**

`pad_and_cat()` (`mwptoolkit.module.Decoder.rnn_decoder.SalignedDecoder` method), 90  
`PAD_ID` (`mwptoolkit.utils.enum_type.EPT` attribute), 145  
`PAD_TOKEN` (`mwptoolkit.utils.enum_type.SpecialTokens` attribute), 147  
`param_search()` (`mwptoolkit.trainer.abstract_trainer.AbstractTrainer` method), 124  
`param_search()` (`mwptoolkit.trainer.supervised_trainer.EPTTrainer` method), 126  
`param_search()` (`mwptoolkit.trainer.supervised_trainer.GTSTrainer` method), 127  
`param_search()` (`mwptoolkit.trainer.supervised_trainer.SupervisedTrainer` method), 134  
`param_search()` (`mwptoolkit.trainer.supervised_trainer.TreeLSTMTrainer` method), 138  
`param_search()` (`mwptoolkit.trainer.supervised_trainer.TRNNTTrainer` method), 136  
`parameters_to_dict()` (`mwptoolkit.data.dataset.abstract_dataset.AbstractDataset` method), 11  
`Parse_Graph_Module` (class in `mwptoolkit.module.Graph.graph_module`), 107  
`play_one()` (`mwptoolkit.module.Layer.tree_layers.DQN` method), 113  
`PLURAL_FORMS` (`mwptoolkit.utils.enum_type.EPT` attribute), 145  
`pos_factor` (`mwptoolkit.module.Decoder.ept_decoder.OpDecoderModel` attribute), 84  
`POS_INF` (`mwptoolkit.utils.enum_type.EPT` attribute), 145  
`PositionalEncoding` (class in `mwptoolkit.module.Embedder.position_embedder`), 97  
`PositionEmbedder` (class in `mwptoolkit.module.Embedder.position_embedder`), 97  
`PositionEmbedder_x` (class in `mwptoolkit.module.Embedder.position_embedder`), 97  
`PositionwiseFeedForward` (class in `mwptoolkit.module.Layer.graph_layers`), 109  
`PositionwiseFeedForward` (class in `mwptoolkit.module.Layer.transformer_layer`), 112  
`Postfix` (`mwptoolkit.utils.enum_type.FixType` attribute), 146  
`postfix_parser()` (in module `mwptoolkit.utils.preprocess_tool.equation_operator`), 150  
`postfix_result()` (`mwptoolkit.evaluate.evaluator.MultiEncDecEvaluator` method), 26  
`postfix_result_multi()` (`mwptoolkit.evaluate.evaluator.MultiEncDecEvaluator` method), 26  
`PostfixEvaluator` (class in `mwptoolkit.evaluate.evaluator`), 28  
`predict()` (`mwptoolkit.model.Graph2Tree.graph2tree.Graph2Tree` method), 62  
`predict()` (`mwptoolkit.model.Graph2Tree.mutiencdec.MultiEncDec` method), 63  
`predict()` (`mwptoolkit.model.PreTrain.bertgen.BERTGen` method), 65  
`predict()` (`mwptoolkit.model.PreTrain.gpt2.GPT2` method), 66  
`predict()` (`mwptoolkit.model.PreTrain.robertagen.RobertaGen` method), 67  
`predict()` (`mwptoolkit.model.Seq2Seq.dns.DNS` method), 38  
`predict()` (`mwptoolkit.model.Seq2Seq.ept.EPT` method), 40  
`predict()` (`mwptoolkit.model.Seq2Seq.groupatt.GroupATT` method), 42  
`predict()` (`mwptoolkit.model.Seq2Seq.mathen.MathEN` method), 43  
`predict()` (`mwptoolkit.model.Seq2Seq.rnnencdec.RNNEncDec`

`method`), 44  
`predict()` (`mwptoolkit.model.Seq2Seq.rnnvae.RNNVAE` `method`), 46  
`predict()` (`mwptoolkit.model.Seq2Seq.saligned.Saligned` `method`), 47  
`predict()` (`mwptoolkit.model.Seq2Seq.transformer.Transformer` `method`), 48  
`predict()` (`mwptoolkit.model.Seq2Tree.berttd.BertTD` `method`), 50  
`predict()` (`mwptoolkit.model.Seq2Tree.gts.GTS` `method`), 51  
`predict()` (`mwptoolkit.model.Seq2Tree.mwpbert.MWPBert` `method`), 52  
`predict()` (`mwptoolkit.model.Seq2Tree.sausolver.SAUSolver` `method`), 54  
`predict()` (`mwptoolkit.model.Seq2Tree.treelstm.TreeLSTM` `method`), 55  
`predict()` (`mwptoolkit.model.Seq2Tree.trnn.TRNN` `method`), 57  
`predict()` (`mwptoolkit.model.Seq2Tree.tsn.TSN` `method`), 58  
`Prediction` (class in `mwptoolkit.module.Layer.tree_layers`), 116  
`PredictModel` (class in `mwptoolkit.module.Decoder.tree_decoder`), 93  
`Prefix` (`mwptoolkit.utils.enum_type.FixType` attribute), 146  
`prefix2tree()` (`mwptoolkit.utils.data_structure.PrefixTree` `method`), 142  
`prefix_result()` (`mwptoolkit.evaluate.evaluator.MultiEncDecEvaluator` `method`), 26  
`prefix_result_multi()` (`mwptoolkit.evaluate.evaluator.MultiEncDecEvaluator` `method`), 27  
`PrefixEvaluator` (class in `mwptoolkit.evaluate.evaluator`), 29  
`PrefixTree` (class in `mwptoolkit.utils.data_structure`), 142  
`PREP_KEY_ANS` (`mwptoolkit.utils.enum_type.EPT` attribute), 145  
`PREP_KEY_EQN` (`mwptoolkit.utils.enum_type.EPT` attribute), 145  
`PREP_KEY_MEM` (`mwptoolkit.utils.enum_type.EPT` attribute), 145  
`preprocess_ept_dataset_()` (in module `mwptoolkit.utils.preprocess_tool.dataset_operator`), 148  
`PretrainDataLoader` (class in `mwptoolkit.data.dataloader.pretrain_data_loader`), 7  
`PretrainDataset` (class in `mwptoolkit.data.dataset.pretrain_dataset`), 17  
`PretrainSeq2SeqTrainer` (class in `mwptoolkit.trainer.supervised_trainer`), 130  
`PretrainTRNNTrainer` (class in `mwptoolkit.trainer.supervised_trainer`), 131  
`problem_level_forward()` (`mwptoolkit.module.Encoder.rnn_encoder.HWCPEncoder` `method`), 101  
`process_gap_encoder_decoder()` (`mwptoolkit.model.Seq2Seq.groupatt.GroupATT` `method`), 42  
`push()` (`mwptoolkit.module.Environment.stack_machine.StackMachine` `method`), 106

## Q

`query()` (`mwptoolkit.utils.data_structure.GoldTree` `method`), 142

## R

`read_ape200k_source()` (in module `mwptoolkit.utils.utils`), 156  
`read_json_data()` (in module `mwptoolkit.utils.utils`), 156  
`read_math23k_source()` (in module `mwptoolkit.utils.utils`), 156  
`read_pos_from_file()` (`mwptoolkit.data.dataset.dataset_multiencdec.DatasetMultiEncDec` `method`), 15  
`RecurCell()` (`mwptoolkit.module.Layer.tree_layers.RecursiveNN` `method`), 117  
`RecursiveNN` (class in `mwptoolkit.module.Layer.tree_layers`), 117  
`refine_formula_as_prefix()` (in module `mwptoolkit.utils.preprocess_tool.dataset_operator`), 148  
`RelevantScore` (class in `mwptoolkit.module.Attention.seq_attention`), 74  
`replace_masked_values()` (in module `mwptoolkit.module.Encoder.graph_based_encoder`), 100  
`required_field` (`mwptoolkit.module.Decoder.ept_decoder.DecoderModel` property), 77  
`required_field` (`mwptoolkit.module.Decoder.ept_decoder.ExpressionPointerTransformer` property), 81  
`required_field` (`mwptoolkit.module.Decoder.ept_decoder.ExpressionTransformer` property), 82  
`required_field` (`mwptoolkit.module.Decoder.ept_decoder.VanillaOpTransformer` property), 86  
`reset()` (`mwptoolkit.loss.abstract_loss.AbstractLoss` `method`), 31

`reset_dataset()` (mwptoolkit.data.dataset.abstract\_dataset.AbstractDataset method), 11  
`reset_parameters()` (mwptoolkit.module.Layer.graph\_layers.GraphConvolution method), 108  
`reset_parameters()` (mwptoolkit.module.Layer.transformer\_layer.TransformerLayer method), 113  
`result()` (mwptoolkit.evaluate.evaluator.AbstractEvaluator method), 25  
`result()` (mwptoolkit.evaluate.evaluator.InfixEvaluator method), 25  
`result()` (mwptoolkit.evaluate.evaluator.MultiEncDecEvaluator method), 27  
`result()` (mwptoolkit.evaluate.evaluator.MultiWayTreeEvaluator method), 28  
`result()` (mwptoolkit.evaluate.evaluator.PostfixEvaluator method), 28  
`result()` (mwptoolkit.evaluate.evaluator.PrefixEvaluator method), 29  
`result_multi()` (mwptoolkit.evaluate.evaluator.AbstractEvaluator method), 25  
`result_multi()` (mwptoolkit.evaluate.evaluator.InfixEvaluator method), 25  
`result_multi()` (mwptoolkit.evaluate.evaluator.MultiEncDecEvaluator method), 27  
`result_multi()` (mwptoolkit.evaluate.evaluator.MultiWayTreeEvaluator method), 28  
`result_multi()` (mwptoolkit.evaluate.evaluator.PostfixEvaluator method), 29  
`result_multi()` (mwptoolkit.evaluate.evaluator.PrefixEvaluator method), 29  
`RNNBasedTreeDecoder` (class in mwptoolkit.module.Decoder.tree\_decoder), 93  
`RNNEncDec` (class in mwptoolkit.model.Seq2Seq.rnnencdec), 43  
`RNNVAE` (class in mwptoolkit.model.Seq2Seq.rnnvae), 45  
`RobertaEmbedder` (class in mwptoolkit.module.Embedder.roberta\_embedder), 98  
`RobertaGen` (class in mwptoolkit.model.PreTrain.robortagen), 66  
`rule1_filter()` (mwptoolkit.model.Seq2Seq.dns.DNS method), 38  
`rule2_filter()` (mwptoolkit.model.Seq2Seq.dns.DNS method), 38  
`rule3_filter()` (mwptoolkit.model.Seq2Seq.dns.DNS method), 38  
`rule4_filter()` (mwptoolkit.model.Seq2Seq.dns.DNS method), 38  
`rule5_filter()` (mwptoolkit.model.Seq2Seq.dns.DNS method), 38  
`rule_filter_()` (mwptoolkit.model.Seq2Seq.dns.DNS method), 38  
`run()` (mwptoolkit.evaluate.evaluator.Solver method), 30  
`run_toolkit()` (in module mwptoolkit.quick\_start), 159

## S

`Saligned` (class in mwptoolkit.model.Seq2Seq.saligned), 46  
`SalignedDecoder` (class in mwptoolkit.module.Decoder.rnn\_decoder), 89  
`SalignedEncoder` (class in mwptoolkit.module.Encoder.rnn\_encoder), 102  
`SalignedTrainer` (class in mwptoolkit.trainer.supervised\_trainer), 132  
`SARTreeDecoder` (class in mwptoolkit.module.Decoder.tree\_decoder), 93  
`SAUSolver` (class in mwptoolkit.model.Seq2Tree.sausolver), 53  
`SAUSolverTrainer` (class in mwptoolkit.trainer.supervised\_trainer), 131  
`save_config()` (mwptoolkit.config.configuration.Config method), 2  
`save_dataset()` (mwptoolkit.data.dataset.abstract\_dataset.AbstractDataset method), 11  
`save_dataset()` (mwptoolkit.data.dataset.dataset\_ept.DatasetEPT method), 13  
`save_dataset()` (mwptoolkit.data.dataset.dataset\_hms.DatasetHMS method), 14  
`save_dataset()` (mwptoolkit.data.dataset.dataset\_multiencdec.DatasetMultiEncDec method), 15  
`save_dataset()` (mwptoolkit.data.dataset.multi\_equation\_dataset.MultiEquationDataset method), 17  
`save_dataset()` (mwptoolkit.data.dataset.pretrain\_dataset.PretrainDataset method), 18  
`save_dataset()` (mwptoolkit.data.dataset.single\_equation\_dataset.SingleEquationDataset method), 20  
`save_dataset()` (mwptoolkit.data.dataset.template\_dataset.TemplateDataset method), 22  
`Score` (class in mwptoolkit.module.Layer.tree\_layers), 117



`score_pn()` (*mwptoolkit.module.Decoder.tree\_decoder.PreSeq2SeqL* forward() (mwptoolkit.model.Seq2Tree.trnn.TRNN method), 93  
`ScoreModel` (class in *mwptoolkit.module.Layer.tree\_layers*), 117  
`seg_and_tag_ape200k()` (in module *mwptoolkit.utils.preprocess\_tool.number\_transfer*), 154  
`seg_and_tag_asdiv_a()` (in module *mwptoolkit.utils.preprocess\_tool.number\_transfer*), 154  
`seg_and_tag_hmwp()` (in module *mwptoolkit.utils.preprocess\_tool.number\_transfer*), 154  
`seg_and_tag_math23k()` (in module *mwptoolkit.utils.preprocess\_tool.number\_transfer*), 154  
`seg_and_tag_mawps()` (in module *mwptoolkit.utils.preprocess\_tool.number\_transfer*), 154  
`seg_and_tag_mawps_single()` (in module *mwptoolkit.utils.preprocess\_tool.number\_transfer*), 154  
`seg_and_tag_multi()` (in module *mwptoolkit.utils.preprocess\_tool.number\_transfer*), 154  
`seg_and_tag_single()` (in module *mwptoolkit.utils.preprocess\_tool.number\_transfer*), 154  
`seg_and_tag_svamp()` (in module *mwptoolkit.utils.preprocess\_tool.number\_transfer*), 154  
`SelfAttention` (class in *mwptoolkit.module.Attention.self\_attention*), 72  
`SelfAttentionMask` (class in *mwptoolkit.module.Attention.self\_attention*), 73  
`SelfAttentionRNNEncoder` (class in *mwptoolkit.module.Encoder.rnn\_encoder*), 102  
`SemanticAlignmentModule` (class in *mwptoolkit.module.Layer.tree\_layers*), 118  
`Semantically_Aligned_Regularization()` (*mwptoolkit.module.Decoder.tree\_decoder.SARTreeDecoder* method), 93  
`sentence2tree()` (*mwptoolkit.utils.data\_structure.DependencyTree* method), 141  
`seq2seq_calculate_loss()` (*mwptoolkit.model.Seq2Tree.trnn.TRNN* method), 57  
`seq2seq_decoder_forward()` (*mwptoolkit.model.Seq2Tree.trnn.TRNN* method), 57  
`seq2seq_encoder_forward()` (*mwptoolkit.model.Seq2Tree.trnn.TRNN* method), 57  
`seq2seq_generate_t()` (*mwptoolkit.model.Seq2Tree.trnn.TRNN* method), 57  
`seq2seq_generate_without_t()` (*mwptoolkit.model.Seq2Tree.trnn.TRNN* method), 57  
`SEQ_END_EQN` (*mwptoolkit.utils.enum\_type.EPT* attribute), 145  
`SEQ_END_EQN_ID` (*mwptoolkit.utils.enum\_type.EPT* attribute), 145  
`SEQ_EQ_SGN_ID` (*mwptoolkit.utils.enum\_type.EPT* attribute), 145  
`SEQ_GEN_NUM_ID` (*mwptoolkit.utils.enum\_type.EPT* attribute), 145  
`SEQ_GEN_VAR_ID` (*mwptoolkit.utils.enum\_type.EPT* attribute), 145  
`SEQ_NEW_EQN` (*mwptoolkit.utils.enum\_type.EPT* attribute), 145  
`SEQ_NEW_EQN_ID` (*mwptoolkit.utils.enum\_type.EPT* attribute), 145  
`SEQ_PTR_NUM` (*mwptoolkit.utils.enum\_type.EPT* attribute), 145  
`SEQ_PTR_NUM_ID` (*mwptoolkit.utils.enum\_type.EPT* attribute), 145  
`SEQ_PTR_TOKENS` (*mwptoolkit.utils.enum\_type.EPT* attribute), 145  
`SEQ_PTR_VAR` (*mwptoolkit.utils.enum\_type.EPT* attribute), 145  
`SEQ_PTR_VAR_ID` (*mwptoolkit.utils.enum\_type.EPT* attribute), 145  
`SEQ_TOKENS` (*mwptoolkit.utils.enum\_type.EPT* attribute), 145  
`SEQ_UNK_TOK` (*mwptoolkit.utils.enum\_type.EPT* attribute), 145  
`SEQ_UNK_TOK_ID` (*mwptoolkit.utils.enum\_type.EPT* attribute), 145  
`SeqAttention` (class in *mwptoolkit.module.Attention.seq\_attention*), 74  
`sequence_mask()` (in module *mwptoolkit.loss.masked\_cross\_entropy\_loss*), 33  
`set_left_node()` (*mwptoolkit.module.Layer.tree\_layers.Node* method), 115  
`set_left_node()` (*mwptoolkit.utils.data\_structure.Node* method), 142  
`set_right_node()` (*mwptoolkit.module.Layer.tree\_layers.Node* method), 115  
`set_right_node()` (*mwptoolkit.utils.data\_structure.Node* method), 142

`toolkit.utils.data_structure.Node` (method), 142  
 `shared_decoder_layer` (mwptoolkit.module.Decoder.ept\_decoder.ExpressionDataLoader attribute), 79  
 `shift_target()` (mwptoolkit.model.Seq2Seq.ept.EPT method), 40  
 `Single` (mwptoolkit.utils.enum\_type.Operators attribute), 147  
 `SingleEquation` (mwptoolkit.utils.enum\_type.TaskType attribute), 147  
 `SingleEquationDataLoader` (class in mwptoolkit.data.data\_loader.single\_equation\_data\_loader), 8  
 `SingleEquationDataset` (class in mwptoolkit.data.dataset.single\_equation\_dataset), 19  
 `SmoothCrossEntropyLoss` (class in mwptoolkit.loss.smoothed\_cross\_entropy\_loss), 34  
 `SmoothedCrossEntropyLoss` (class in mwptoolkit.loss.smoothed\_cross\_entropy\_loss), 34  
 `soft_cross_entropy_loss()` (in module mwptoolkit.model.Seq2Tree.tsn), 60  
 `soft_target_loss()` (in module mwptoolkit.model.Seq2Tree.tsn), 60  
 `softmax` (mwptoolkit.module.Decoder.ept\_decoder.VanillaOpTransform attribute), 86  
 `Solver` (class in mwptoolkit.evaluate.evaluator), 30  
 `SOS_TOKEN` (mwptoolkit.utils.enum\_type.SpecialTokens attribute), 147  
 `span_level_deprel_tree_to_file()` (in module mwptoolkit.utils.preprocess\_tool.sentence\_operator), 155  
 `SpecialTokens` (class in mwptoolkit.utils.enum\_type), 147  
 `SPIECE_UNDERLINE` (mwptoolkit.utils.enum\_type.EPT attribute), 145  
 `split_number()` (in module mwptoolkit.utils.preprocess\_tool.number\_operator), 152  
 `split_sentence()` (in module mwptoolkit.utils.preprocess\_tool.sentence\_operator), 155  
 `Squeeze` (class in mwptoolkit.module.Decoder.ept\_decoder), 84  
 `src_to_mask()` (in module mwptoolkit.module.Attention.group\_attention), 70  
 `src_to_mask()` (mwptoolkit.module.Attention.group\_attention.GroupAttention method), 69  
 `StackMachine` (class in mwptoolkit.module.Environment.stack\_machine), 104  
 `start()` (mwptoolkit.module.Strategy.beam\_search.Beam\_Search\_Hypothesis method), 120  
 `stop()` (mwptoolkit.module.Strategy.beam\_search.Beam\_Search\_Hypothesis method), 120  
 `str2float()` (in module mwptoolkit.utils.utils), 156  
 `student_calculate_loss()` (mwptoolkit.model.Seq2Tree.tsn.TSN method), 58  
 `student_net_1_decoder_forward()` (mwptoolkit.model.Seq2Tree.tsn.TSN method), 58  
 `student_net_2_decoder_forward()` (mwptoolkit.model.Seq2Tree.tsn.TSN method), 59  
 `student_net_decoder_forward()` (mwptoolkit.model.Seq2Tree.tsn.TSN method), 59  
 `student_net_encoder_forward()` (mwptoolkit.model.Seq2Tree.tsn.TSN method), 59  
 `student_net_forward()` (mwptoolkit.model.Seq2Tree.tsn.TSN method), 59  
 `student_test()` (mwptoolkit.model.Seq2Tree.tsn.TSN method), 59  
 `SublayerConnection` (class in mwptoolkit.module.Layer.transformer\_layer), 112  
 `Submodule_types()` (in module mwptoolkit.model.Seq2Seq.ept), 40  
 `SubTreeMerger` (class in mwptoolkit.module.Layer.tree\_layers), 118  
 `SupervisedTrainer` (class in mwptoolkit.trainer.supervised\_trainer), 133  
 `SupervisingMode` (class in mwptoolkit.utils.enum\_type), 147  
 `SVAMP` (mwptoolkit.utils.enum\_type.DatasetName attribute), 142  
 `symbol2idx()` (mwptoolkit.model.Seq2Tree.trnn.TRNN method), 57  

## T

 `TaskType` (class in mwptoolkit.utils.enum\_type), 147  
 `teacher_calculate_loss()` (mwptoolkit.model.Seq2Tree.tsn.TSN method), 59  
 `teacher_net_decoder_forward()` (mwptoolkit.model.Seq2Tree.tsn.TSN method), 59  
 `teacher_net_encoder_forward()` (mwptoolkit.model.Seq2Tree.tsn.TSN method), 59

60  
 teacher\_net\_forward() (mwptoolkit.model.Seq2Tree.tsn.TSN method), 60  
 teacher\_test() (mwptoolkit.model.Seq2Tree.tsn.TSN method), 60  
 template2tree() (mwptoolkit.model.Seq2Tree.trnn.TRNN method), 57  
 TemplateDataLoader (class in mwptoolkit.data.data\_loader.template\_data\_loader), 9  
 TemplateDataset (class in mwptoolkit.data.dataset.template\_dataset), 20  
 TemplateTrainer (class in mwptoolkit.trainer.template\_trainer), 139  
 Test (mwptoolkit.utils.enum\_type.DatasetType attribute), 143  
 test() (mwptoolkit.module.Layer.tree\_layers.RecursiveNN method), 117  
 test() (mwptoolkit.trainer.abstract\_trainer.AbstractTrainer method), 124  
 test() (mwptoolkit.trainer.supervised\_trainer.EPTTrainer method), 126  
 test() (mwptoolkit.trainer.supervised\_trainer.GTSTrainer method), 127  
 test() (mwptoolkit.trainer.supervised\_trainer.SalignedTrainer method), 133  
 test() (mwptoolkit.trainer.supervised\_trainer.SupervisedTrainer method), 134  
 test() (mwptoolkit.trainer.supervised\_trainer.TreeLSTMTrainer method), 138  
 test() (mwptoolkit.trainer.supervised\_trainer.TRNNTrainer method), 136  
 test() (mwptoolkit.trainer.supervised\_trainer.TSNTrainer method), 137  
 test() (mwptoolkit.trainer.template\_trainer.TemplateTrainer method), 139  
 test\_traverse() (mwptoolkit.module.Layer.tree\_layers.RecursiveNN method), 117  
 test\_with\_cross\_validation() (in module mwptoolkit.quick\_start), 159  
 test\_with\_train\_valid\_test\_split() (in module mwptoolkit.quick\_start), 159  
 time\_since() (in module mwptoolkit.utils.utils), 156  
 to\_dict() (mwptoolkit.config.configuration.Config method), 2  
 to\_list() (mwptoolkit.utils.data\_structure.Tree method), 142  
 to\_string() (mwptoolkit.utils.data\_structure.Tree method), 142  
 token\_resize() (mwptoolkit.module.Embedder.bert\_embedder.BertEmbedder method), 95  
 token\_resize() (mwptoolkit.module.Embedder.roberta\_embedder.RobertaEmbedder method), 98  
 token\_resize() (mwptoolkit.module.Encoder.transformer\_encoder.BertEncoder method), 103  
 TOP\_LEVEL\_CLASSES (mwptoolkit.utils.enum\_type.EPT attribute), 145  
 topk\_sampling() (in module mwptoolkit.module.Strategy.sampling), 121  
 Train (mwptoolkit.utils.enum\_type.DatasetType attribute), 143  
 train\_cross\_validation() (in module mwptoolkit.quick\_start), 159  
 train\_process() (in module mwptoolkit.hyper\_search), 157  
 train\_tree() (mwptoolkit.model.Seq2Tree.sausolver.SAUSolver method), 54  
 train\_with\_cross\_validation() (in module mwptoolkit.quick\_start), 159  
 train\_with\_train\_valid\_test\_split() (in module mwptoolkit.quick\_start), 159  
 training (mwptoolkit.loss.smoothed\_cross\_entropy\_loss.SmoothedCrossEntropyLoss attribute), 35  
 training (mwptoolkit.model.Graph2Tree.graph2tree.Graph2Tree attribute), 62  
 training (mwptoolkit.model.Graph2Tree.multienccdec.MultiEncDec attribute), 63  
 training (mwptoolkit.model.PreTrain.bertgen.BERTGen attribute), 65  
 training (mwptoolkit.model.PreTrain.gpt2.GPT2 attribute), 66  
 training (mwptoolkit.model.PreTrain.robortagen.RobertaGen attribute), 67  
 training (mwptoolkit.model.Seq2Seq.dns.DNS attribute), 38  
 training (mwptoolkit.model.Seq2Seq.ept.EPT attribute), 40  
 training (mwptoolkit.model.Seq2Seq.groupatt.GroupATT attribute), 42  
 training (mwptoolkit.model.Seq2Seq.mathen.MathEN attribute), 43  
 training (mwptoolkit.model.Seq2Seq.rnnencdec.RNNEncDec attribute), 44  
 training (mwptoolkit.model.Seq2Seq.rnnvae.RNNVAE attribute), 46  
 training (mwptoolkit.model.Seq2Seq.saligned.Saligned attribute), 47  
 training (mwptoolkit.model.Seq2Seq.transformer.Transformer attribute), 48  
 training (mwptoolkit.model.Seq2Tree.berttd.BertTD attribute), 50  
 training (mwptoolkit.model.Seq2Tree.gts.GTS attribute), 51





attribute), 104  
 training (mwptoolkit.module.Graph.gcn.GCN attribute), 106  
 training (mwptoolkit.module.Graph.graph\_module.GraphModule attribute), 107  
 training (mwptoolkit.module.Graph.graph\_module.Num\_GrainModule attribute), 107  
 training (mwptoolkit.module.Graph.graph\_module.Parse\_GrainModule attribute), 107  
 training (mwptoolkit.module.Layer.graph\_layers.GraphConvolution attribute), 108  
 training (mwptoolkit.module.Layer.graph\_layers.LayerNorm attribute), 108  
 training (mwptoolkit.module.Layer.graph\_layers.MeanAggregation attribute), 109  
 training (mwptoolkit.module.Layer.graph\_layers.PositionwiseFeedForward attribute), 109  
 training (mwptoolkit.module.Layer.layers.GenVar attribute), 110  
 training (mwptoolkit.module.Layer.layers.Transformer attribute), 110  
 training (mwptoolkit.module.Layer.layers.TreeAttnDecoder attribute), 110  
 training (mwptoolkit.module.Layer.transformer\_layer.EPTTransformerDecoder attribute), 111  
 training (mwptoolkit.module.Layer.transformer\_layer.GAEncoderLayer attribute), 111  
 training (mwptoolkit.module.Layer.transformer\_layer.LayerNorm attribute), 112  
 training (mwptoolkit.module.Layer.transformer\_layer.PositionwiseFeedForward attribute), 112  
 training (mwptoolkit.module.Layer.transformer\_layer.SublayerConnection attribute), 112  
 training (mwptoolkit.module.Layer.transformer\_layer.TransformerLayer attribute), 113  
 training (mwptoolkit.module.Layer.tree\_layers.Dec\_LSTM attribute), 114  
 training (mwptoolkit.module.Layer.tree\_layers.Decomposition attribute), 114  
 training (mwptoolkit.module.Layer.tree\_layers.DQN attribute), 113  
 training (mwptoolkit.module.Layer.tree\_layers.GateNN attribute), 114  
 training (mwptoolkit.module.Layer.tree\_layers.GenerateNode attribute), 115  
 training (mwptoolkit.module.Layer.tree\_layers.Merge attribute), 115  
 training (mwptoolkit.module.Layer.tree\_layers.NodeEmbeddingLayer attribute), 116  
 training (mwptoolkit.module.Layer.tree\_layers.NodeGenerator attribute), 116  
 training (mwptoolkit.module.Layer.tree\_layers.Prediction attribute), 117  
 training (mwptoolkit.module.Layer.tree\_layers.RecursiveNN attribute), 117  
 training (mwptoolkit.module.Layer.tree\_layers.Score attribute), 117  
 training (mwptoolkit.module.Layer.tree\_layers.ScoreModel attribute), 118  
 training (mwptoolkit.module.Layer.tree\_layers.SemanticAlignmentModule attribute), 118  
 training (mwptoolkit.module.Layer.tree\_layers.SubTreeMerger attribute), 118  
 training (mwptoolkit.module.Layer.tree\_layers.TreeAttention attribute), 119  
 training (mwptoolkit.module.Layer.tree\_layers.TreeEmbeddingModel attribute), 119  
 trans\_symbol\_2\_number() (in module mwptoolkit.utils.preprocess\_tool.equation\_operator), 151  
 trans\_symbol\_2\_number() (in module mwptoolkit.utils.preprocess\_tool.number\_operator), 152  
 Transformer (class in mwptoolkit.model.Seq2Seq.transformer), 47  
 Transformer (class in mwptoolkit.module.Layer.layers), 110  
 TransformerDecoder (class in mwptoolkit.module.Decoder.transformer\_decoder), 109  
 TransformerEncoder (class in mwptoolkit.module.Encoder.transformer\_encoder), 104  
 TransformerLayer (class in mwptoolkit.module.Layer.transformer\_layer), 113  
 traverse() (mwptoolkit.module.Layer.tree\_layers.RecursiveNN method), 117  
 Tree (class in mwptoolkit.utils.data\_structure), 142  
 tree2equ() (mwptoolkit.utils.data\_structure.AbstractTree method), 141  
 tree2equ() (mwptoolkit.utils.data\_structure.BinaryTree method), 141  
 tree2equation() (mwptoolkit.model.Seq2Tree.trnn.TRNN method), 57  
 tree\_decoder\_forward() (mwptoolkit.model.Graph2Tree.multienccdec.MultiEncDec method), 63  
 TreeAttention (class in mwptoolkit.module.Attention.tree\_attention), 115  
 TreeAttention (class in mwptoolkit.module.Layer.tree\_layers), 118  
 TreeAttnDecoderRNN (class in mwptoolkit.module.Layer.layers), 110  
 TreeBeam (class in mwptoolkit.module.Strategy.beam\_search), 120

`TreeDecoder` (class in `mwptoolkit.module.Decoder.tree_decoder`), 94  
`TreeEmbedding` (class in `mwptoolkit.module.Layer.tree_layers`), 119  
`TreeEmbeddingModel` (class in `mwptoolkit.module.Layer.tree_layers`), 119  
`TreeLSTM` (class in `mwptoolkit.model.Seq2Tree.treelstm`), 54  
`TreeLSTMTrainer` (class in `mwptoolkit.trainer.supervised_trainer`), 137  
`TreeNode` (class in `mwptoolkit.module.Layer.tree_layers`), 119  
`TRNN` (class in `mwptoolkit.model.Seq2Tree.trnn`), 56  
`TRNNTrainer` (class in `mwptoolkit.trainer.supervised_trainer`), 135  
`TSN` (class in `mwptoolkit.model.Seq2Tree.tsn`), 57  
`TSNTrainer` (class in `mwptoolkit.trainer.supervised_trainer`), 136

## U

`UNK_TOKEN` (`mwptoolkit.utils.enum_type.SpecialTokens` attribute), 147

## V

`Valid` (`mwptoolkit.utils.enum_type.DatasetType` attribute), 143  
`VanillaOpTransformer` (class in `mwptoolkit.module.Decoder.ept_decoder`), 85  
`VAR_MAX` (`mwptoolkit.utils.enum_type.EPT` attribute), 146  
`VAR_PREFIX` (`mwptoolkit.utils.enum_type.EPT` attribute), 146

## W

`weakly_supervised` (`mwptoolkit.utils.enum_type.SupervisingMode` attribute), 147  
`word_level_forward()` (`mwptoolkit.module.Encoder.rnn_encoder.HWCPDecoder` method), 101  
`write_json_data()` (in module `mwptoolkit.utils.utils`), 156

## Z

`zh` (`mwptoolkit.utils.enum_type.DatasetLanguage` attribute), 142