fastNLP Documentation

Release 0.2

xpqiu

Feb 04, 2019

Contents

A Modularized and Extensible Toolkit for Natural Language Processing. Currently still in incubation.

CHAPTER 1

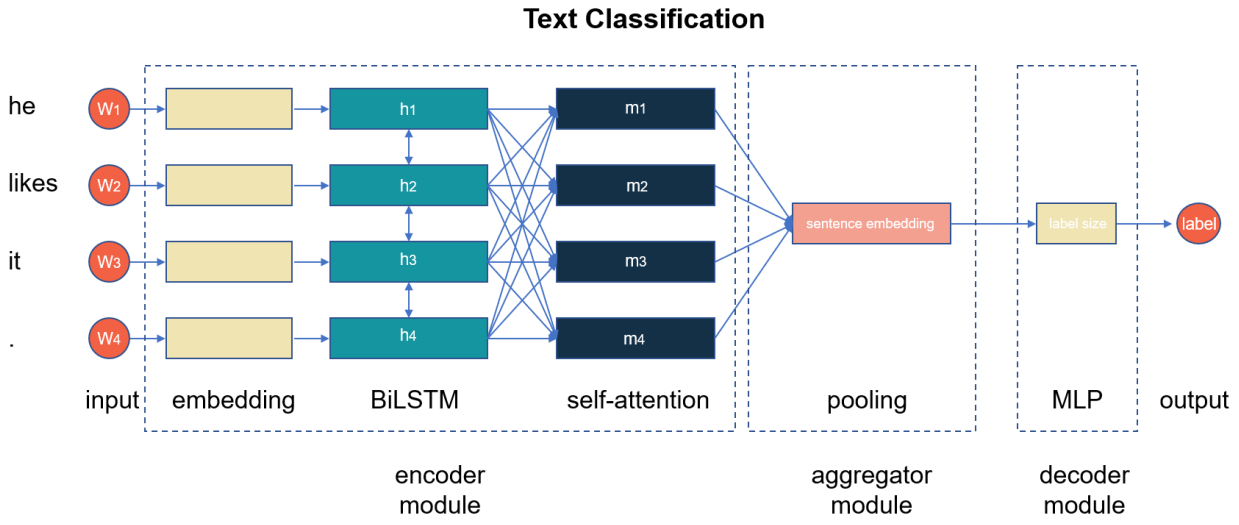
Introduction

FastNLP is a modular Natural Language Processing system based on PyTorch, built for fast development of NLP models.

A deep learning NLP model is the composition of three types of modules:

module type	functionality	example
encoder	encode the input into some abstract representation	embedding, RNN, CNN, transformer
aggregator	aggregate and reduce information	self-attention, max-pooling
decoder	decode the representation into the output	MLP, CRF

For example:



CHAPTER 2

User's Guide

2.1 Installation

Make sure your environment satisfies https://github.com/fastnlp/fastNLP/blob/master/requirements.txt .

Run the following commands to install fastNLP package:

```
pip install fastNLP
```

2.2 Quickstart

2.2.1 FastNLP 1

https://github.com/fastnlp/fastNLP/blob/master/tutorials/fastnlp_1min_tutorial.ipynb

step 1

```
from fastNLP import DataSet
# linux_path = "../test/data_for_tests/tutorial_sample_dataset.csv"
win_path = "C:\\Users\zyfeng\Desktop\FudanNLP\\fastNLP\\test\\data_for_
↪tests\\tutorial_sample_dataset.csv"
ds = DataSet.read_csv(win_path, headers=('raw_sentence', 'label'), sep='\t')
```

step 2

1. 2. 3.

```
#
ds.apply(lambda x: x['raw_sentence'].lower(), new_field_name='raw_sentence')
# labelint
ds.apply(lambda x: int(x['label']), new_field_name='label_seq', is_target=True)

def split_sent(ins):
    return ins['raw_sentence'].split()
ds.apply(split_sent, new_field_name='words', is_input=True)
```

```
# /
train_data, dev_data = ds.split(0.3)
print("Train size: ", len(train_data))
print("Test size: ", len(dev_data))
```

```
Train size:  54
Test size:  23
```

```
from fastNLP import Vocabulary
vocab = Vocabulary(min_freq=2)
train_data.apply(lambda x: [vocab.add(word) for word in x['words']])

# index, Vocabulary.to_index(word)
train_data.apply(lambda x: [vocab.to_index(word) for word in x['words']], new_field_
↪name='word_seq', is_input=True)
dev_data.apply(lambda x: [vocab.to_index(word) for word in x['words']], new_field_
↪name='word_seq', is_input=True)
```

step 3

```
from fastNLP.models import CNNText
model = CNNText(embed_num=len(vocab), embed_dim=50, num_classes=5, padding=2,␣
↪dropout=0.1)
```

step 4

```
from fastNLP import Trainer, CrossEntropyLoss, AccuracyMetric
trainer = Trainer(model=model,
                  train_data=train_data,
                  dev_data=dev_data,
                  loss=CrossEntropyLoss(),
                  metrics=AccuracyMetric()
                  )
trainer.train()
print('Train finished!')
```

```
training epochs started 2018-12-07 14:03:41
```

```
HBox(children=(IntProgress(value=0, layout=Layout(flex='2'), max=6), HTML(value='')),␣
↪layout=Layout(display='i...
```

```
Epoch 1/3. Step:2/6. AccuracyMetric: acc=0.26087
Epoch 2/3. Step:4/6. AccuracyMetric: acc=0.347826
Epoch 3/3. Step:6/6. AccuracyMetric: acc=0.608696
Train finished!
```

2.2.2 fastNLP 10

https://github.com/fastnlp/fastNLP/blob/master/tutorials/fastnlp_10min_tutorial.ipynb

fastNLP

DataSet & Instance

fastNLPDataSetInstanceDataSetInstanceDataSetInstanceInstance

read_*DataSet

```
from fastNLP import DataSet
from fastNLP import Instance

# csvDataSet
win_path = "C:\\Users\zyfeng\Desktop\FudanNLP\\fastNLP\\test\\data_for_
↪tests\\tutorial_sample_dataset.csv"
dataset = DataSet.read_csv(win_path, headers=('raw_sentence', 'label'), sep='\t')
print(dataset[0])
```

```
{'raw_sentence': A series of escapades demonstrating the adage that what is good for␣
↪the goose is also good for the gander , some of which occasionally amuses but none␣
↪of which amounts to much of a story .,
'label': 1}
```

```
# DataSet.append(Instance)

dataset.append(Instance(raw_sentence='fake data', label='0'))
dataset[-1]
```

```
{'raw_sentence': fake data,
'label': 0}
```

```
# DataSet.apply(func, new_field_name)

#
dataset.apply(lambda x: x['raw_sentence'].lower(), new_field_name='raw_sentence')
# labelint
dataset.apply(lambda x: int(x['label']), new_field_name='label_seq', is_target=True)
#
dataset.drop(lambda x: len(x['raw_sentence'].split()) == 0)
def split_sent(ins):
    return ins['raw_sentence'].split()
dataset.apply(split_sent, new_field_name='words', is_input=True)
```

```
# DataSet.drop(func)
#
dataset.drop(lambda x: len(x['words']) <= 3)
```

```
#

test_data, train_data = dataset.split(0.3)
print("Train size: ", len(test_data))
print("Test size: ", len(train_data))
```

```
Train size:  54
Test size:
```

Vocabulary

fastNLPVocabulary

```
from fastNLP import Vocabulary

# , Vocabulary.add(word)
vocab = Vocabulary(min_freq=2)
train_data.apply(lambda x: [vocab.add(word) for word in x['words']])
vocab.build_vocab()

# index, Vocabulary.to_index(word)
train_data.apply(lambda x: [vocab.to_index(word) for word in x['words']], new_field_
↪name='word_seq', is_input=True)
test_data.apply(lambda x: [vocab.to_index(word) for word in x['words']], new_field_
↪name='word_seq', is_input=True)


print(test_data[0])
```

```
{'raw_sentence': the plot is romantic comedy boilerplate from start to finish .,
'label': 2,
'label_seq': 2,
'words': ['the', 'plot', 'is', 'romantic', 'comedy', 'boilerplate', 'from', 'start',
↪'to', 'finish', '.'],
'word_seq': [2, 13, 9, 24, 25, 26, 15, 27, 11, 28, 3]}
```

```
# gandataset
from fastNLP.core.batch import Batch
from fastNLP.core.sampler import RandomSampler

batch_iterator = Batch(dataset=train_data, batch_size=2, sampler=RandomSampler())
for batch_x, batch_y in batch_iterator:
    print("batch_x has: ", batch_x)
    print("batch_y has: ", batch_y)
    break
```

```
batch_x has:  {'words': array([list(['this', 'kind', 'of', 'hands-on', 'storytelling',
↪ 'is', 'ultimately', 'what', 'makes', 'shanghai', 'ghetto', 'move', 'beyond', 'a',
↪'good', ',', 'dry', ',', 'reliable', 'textbook', 'and', 'what', 'allows', 'it', 'to
↪', 'rank', 'with', 'its', 'worthy', 'predecessors', '.']),
```

(continues on next page)

(continued from previous page)

```
       list(['the', 'entire', 'movie', 'is', 'filled', 'with', 'deja', 'vu', 'moments
↪', '.'])],
      dtype=object), 'word_seq': tensor([[  19,  184,    6,    1,  481,    9,  206,
↪50,   91, 1210, 1609, 1330,
          495,    5,   63,    4, 1269,    4,    1, 1184,    7,   50, 1050,   10,
            8, 1611,   16,   21, 1039,    1,    2],
        [   3,  711,   22,    9, 1282,   16, 2482, 2483,  200,    2,    0,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
            0,    0,    0,    0,    0,    0,    0]])}
batch_y has:  {'label_seq': tensor([3, 2])}
```

Model

```
# Pytorch

from fastNLP.models import CNNText
model = CNNText(embed_num=len(vocab), embed_dim=50, num_classes=5, padding=2,
↪dropout=0.1)
model
```

```
CNNText(
  (embed): Embedding(
    (embed): Embedding(77, 50, padding_idx=0)
    (dropout): Dropout(p=0.0)
  )
  (conv_pool): ConvMaxpool(
    (convs): ModuleList(
      (0): Conv1d(50, 3, kernel_size=(3,), stride=(1,), padding=(2,))
      (1): Conv1d(50, 4, kernel_size=(4,), stride=(1,), padding=(2,))
      (2): Conv1d(50, 5, kernel_size=(5,), stride=(1,), padding=(2,))
    )
  )
  (dropout): Dropout(p=0.1)
  (fc): Linear(
    (linear): Linear(in_features=12, out_features=5, bias=True)
  )
)
```

Trainer & Tester

fastNLPTrainer

```
from fastNLP import Trainer
from copy import deepcopy
from fastNLP import CrossEntropyLoss
from fastNLP import AccuracyMetric
```

```
# overfitting
copy_model = deepcopy(model)
overfit_trainer = Trainer(model=copy_model,
                          train_data=test_data,
                          dev_data=test_data,
```

(continues on next page)

(continued from previous page)

```
                          loss=CrossEntropyLoss(pred="output", target="label_seq"),
                          metrics=AccuracyMetric(),
                          n_epochs=10,
                          save_path=None)
overfit_trainer.train()
```

```
training epochs started 2018-12-07 14:07:20
```

```
HBox(children=(IntProgress(value=0, layout=Layout(flex='2'), max=20), HTML(value='')),
↪ layout=Layout(display='...
```

```
Epoch 1/10. Step:2/20. AccuracyMetric: acc=0.037037
Epoch 2/10. Step:4/20. AccuracyMetric: acc=0.296296
Epoch 3/10. Step:6/20. AccuracyMetric: acc=0.333333
Epoch 4/10. Step:8/20. AccuracyMetric: acc=0.555556
Epoch 5/10. Step:10/20. AccuracyMetric: acc=0.611111
Epoch 6/10. Step:12/20. AccuracyMetric: acc=0.481481
Epoch 7/10. Step:14/20. AccuracyMetric: acc=0.62963
Epoch 8/10. Step:16/20. AccuracyMetric: acc=0.685185
Epoch 9/10. Step:18/20. AccuracyMetric: acc=0.722222
Epoch 10/10. Step:20/20. AccuracyMetric: acc=0.777778
```

```
# Trainer
trainer = Trainer(model=model,
                  train_data=train_data,
                  dev_data=test_data,
                  loss=CrossEntropyLoss(pred="output", target="label_seq"),
                  metrics=AccuracyMetric(),
                  n_epochs=5)
trainer.train()
print('Train finished!')
```

```
training epochs started 2018-12-07 14:08:10
```

```
HBox(children=(IntProgress(value=0, layout=Layout(flex='2'), max=5), HTML(value='')),␣
↪layout=Layout(display='i...
```

```
Epoch 1/5. Step:1/5. AccuracyMetric: acc=0.037037
Epoch 2/5. Step:2/5. AccuracyMetric: acc=0.037037
Epoch 3/5. Step:3/5. AccuracyMetric: acc=0.037037
Epoch 4/5. Step:4/5. AccuracyMetric: acc=0.185185
Epoch 5/5. Step:5/5. AccuracyMetric: acc=0.240741
Train finished!
```

```
from fastNLP import Tester

tester = Tester(data=test_data, model=model, metrics=AccuracyMetric())
acc = tester.test()
```

```
[tester]
AccuracyMetric: acc=0.240741
```

In summary

fastNLP Trainer

1. DataSetDataSetfields

```
['raw_sentence', 'word_seq1', 'word_seq2', 'raw_label','label']

    DataSet.set_input('word_seq1', word_seq2', flag=True)'word_seq1', 'word_seq2'input

    DataSet.set_target('label', flag=True)'label'target
```

2.

```
class Model(nn.Module):
    def __init__(self):
        xxx
    def forward(self, word_seq1, word_seq2):
        # (1) DataSetinput field,
        # (2) input field
        xxxx
        # dict
```

3. Trainer

```
(1) DataSetbatch_sizebatchModel.forward
(2)  Model.forward  targetfield Losser
       Model.forwardoutputdictkey{'pred':xxx}, {'output': xxx};
       target, label, target
    loss
       CrossEntropyLosser(prediction, target)forwardoutput{'output': xxx}; 'label
↪'target
       losserCrossEntropyLosser(prediction='output', target='label')
 (3) Metric
     Metric forward  targetfield
```

.

1. DataSetinputtarget

```
inputtargettrain
(1.1) inputfieldModel.forward
(1.2) lossermetric:
        (a)Model.forwardoutput
        (b)targetfield
```

2. forwadDataSetfield

```
(1.1)
 :
     DataSetxseq_lensinputforward
     def forward(self, x, seq_lens):
         pass
     field
```

1. DataSet

2. applyDataSet

```
(2.1) fieldinputfieldtarget
```

3.

```
(3.1) forwardDataSetinputfield
:
    DataSetxseq_lensinputforward
    def forward(self, x, seq_lens):
        pass
    field
(3.2) forwardoutputdict
    {"pred": xx}.
```

2.2.3 fastNLP

https://github.com/fastnlp/fastNLP/blob/master/tutorials/fastnlp_advanced_tutorial/advance_tutorial.ipynb

2.2.4 fastNLP

https://github.com/fastnlp/fastNLP/blob/master/tutorials/tutorial_for_developer.md

CHAPTER 3

API Reference

If you are looking for information on a specific function, class or method, this part of the documentation is for you.

3.1 fastNLP

3.1.1 fastNLP.api

fastNLP.api.api

class fastNLP.api.api.**POS**(*model_path=None*, *device='cpu'*)

FastNLP API for Part-Of-Speech tagging.

Parameters

- **model_path** (*str*) – the path to the model.
- **device** (*str*) – device name such as "cpu" or "cuda:0". Use the same notation as PyTorch.

predict(*content*)

Parameters **content** – list of list of str. Each string is a token(word).

Return answer list of list of str. Each string is a tag.

test(*file_path*)

Test performance over the given data set.

Parameters **file_path** (*str*) –

Returns a dictionary of metric values

fastNLP.api.converter

fastNLP.api.model_zoo

fastNLP.api.pipeline

class fastNLP.api.pipeline.**Pipeline**(*processors=None*)
Pipeline takes a DataSet object as input, runs multiple processors sequentially, and outputs a DataSet object.

fastNLP.api.processor

class fastNLP.api.processor.**FullSpaceToHalfSpaceProcessor**(*field_name*, *change_alpha=True*, *change_digit=True*, *change_punctuation=True*, *change_space=True*)

class fastNLP.api.processor.**Index2WordProcessor**(*vocab*, *field_name*, *new_added_field_name*)
DataSetindexfieldvocabstr

class fastNLP.api.processor.**IndexerProcessor**(*vocab*, *field_name*, *new_added_field_name*, *delete_old_field=False*, *is_input=True*)

vocabulary , fieldindexfieldlist [‘’, ‘’, xxx]

class fastNLP.api.processor.**Num2TagProcessor**(*tag*, *field_name*, *new_added_field_name=None*)
tag

class fastNLP.api.processor.**PreAppendProcessor**(*data*, *field_name*, *new_added_field_name=None*)

fielddata(str)fieldlistfield [data] + instance[field_name]

class fastNLP.api.processor.**SeqLenProcessor**(*field_name*, *new_added_field_name='seq_lens'*, *is_input=True*)
fieldsequence lengthfieldfield

class fastNLP.api.processor.**SliceProcessor**(*start*, *end*, *step*, *field_name*, *new_added_field_name=None*)
fieldinstance[field_name][start:end:step]

class fastNLP.api.processor.**VocabIndexerProcessor**(*field_name*, *new_added_filed_name=None*, *min_freq=1*, *max_size=None*, *verbose=0*, *is_input=True*)

DataSetVocabularyindexindexfieldnew_added_filed_name, new_added_field_name, field_name.

construct_vocab(**datasets*)
DataSetvocabulary

Parameters **datasets** – DataSetvocabulary

Returns

process(**datasets*, *only_index_dataset=None*)

VocabularydatasetDataSetvocabularyvocabularyvocabularyvocabulary index dataset-sonly_index_dataset

Parameters

- **datasets** – DataSet
- **only_index_dataset** – DataSet, or list of DataSet. indexvocabulary

Returns

set_verbose(*verbose*)
processor verbose

Parameters **verbose** – int, 01vocab

Returns

class fastNLP.api.processor.**VocabProcessor**(*field_name*, *min_freq=1*, *max_size=None*)
DataSetvocabulary

3.1.2 fastNLP.core

fastNLP.core.batch

class fastNLP.core.batch.**Batch**(*dataset*, *batch_size*, *sampler=<fastNLP.core.sampler.RandomSampler object>*, *as_numpy=False*, *prefetch=False*)
Batch is an iterable object which iterates over mini-batches.

Example:

```
for batch_x, batch_y in Batch(data_set, batch_size=16,␣
↪sampler=SequentialSampler()):
    # ...
```

Parameters

- **dataset** (DataSet) – a DataSet object
- **batch_size** (*int*) – the size of the batch
- **sampler** (*Sampler*) – a Sampler object
- **as_numpy** (*bool*) – If True, return Numpy array. Otherwise, return torch tensors.
- **prefetch** (*bool*) – If True, use multiprocessing to fetch next batch when training.
- **or torch.device device** (*str*) – the batch's device, if as_numpy is True, device is ignored.

fastNLP.core.dataset

class fastNLP.core.dataset.**DataSet**(*data=None*)
DataSet is the collection of examples. DataSet provides instance-level interface. You can append and access an instance of the DataSet. However, it stores data in a different way: Field-first, Instance-second.

add_field(*name*, *fields*, *padder=<fastNLP.core.fieldarray.AutoPadder object>*, *is_input=False*, *is_target=False*)
Add a new field to the DataSet.

Parameters

- **name** (*str*) – the name of the field.
- **fields** – a list of int, float, or other objects.
- **padder** (*int*) – PadBaseFieldpadding
- **is_input** (*bool*) – whether this field is model input.

- **is_target** (*bool*) – whether this field is label or target.

append(*ins*)
Add an instance to the DataSet. If the DataSet is not empty, the instance must have the same field names as the rest instances in the DataSet.

Parameters **ins** – an Instance object

apply(*func*, *new_field_name=None*, ***kwargs*)
Apply a function to every instance of the DataSet.

Parameters

- **func** – a function that takes an instance as input.
- **new_field_name** (*str*) – If not None, results of the function will be stored as a new field.
- ****kwargs** – Accept parameters will be (1) is_input: boolean, will be ignored if new_field is None. If True, the new field will be as input. (2) is_target: boolean, will be ignored if new_field is None. If True, the new field will be as target.

Return results if new_field_name is not passed, returned values of the function over all instances.

delete_field(*name*)
Delete a field based on the field name.

Parameters **name** – the name of the field to be deleted.

drop(*func*)
Drop instances if a condition holds.

Parameters **func** – a function that takes an Instance object as input, and returns bool. The instance will be dropped if the function returns True.

get_all_fields()
Return all the fields with their names.

Return field_arrays the internal data structure of DataSet.

get_input_name()
Get all field names with *is_input* as True.

Return field_names a list of str

get_length()
Fetch the length of the dataset.

Return length

get_target_name()
Get all field names with *is_target* as True.

Return field_names a list of str

static load(*path*)
Load a DataSet object from pickle.

Parameters **path** (*str*) – the path to the pickle

Return data_set

classmethod read_csv(*csv_path*, *headers=None*, *sep=', '*, *dropna=True*)
Load data from a CSV file and return a DataSet object.

Parameters

- **csv_path** (*str*) – path to the CSV file
- **or Tuple[str] headers** (*List[str]*) – headers of the CSV file
- **sep** (*str*) – delimiter in CSV file. Default: ","
- **dropna** (*bool*) – If True, drop rows that have less entries than headers.

Return dataset the read data set

rename_field(*old_name*, *new_name*)
Rename a field.

Parameters

- **old_name** (*str*) –
- **new_name** (*str*) –

save(*path*)
Save the DataSet object as pickle.

Parameters **path** (*str*) – the path to the pickle

set_input(**field_name*, *flag=True*)
Set the input flag of these fields.

Parameters

- **field_name** – a sequence of str, indicating field names.
- **flag** (*bool*) – Set these fields as input if True. Unset them if False.

set_pad_val(*field_name*, *pad_val*) **Parameters**

- **field_name** – strfieldpad_val
- **pad_val** – intfieldpadderpad_valpadding index

Returns

set_padder(*field_name*, *padder*)
field_namepadder :param field_name: str, fieldpaddingpadder :param padder: PadderBaseNone. Nonepadderfieldpadding. :return:

set_target(**field_names*, *flag=True*)
Change the target flag of these fields.

Parameters

- **field_names** – a sequence of str, indicating field names
- **flag** (*bool*) – Set these fields as target if True. Unset them if False.

split(*dev_ratio*)
Split the dataset into training and development(validation) set.

Parameters **dev_ratio** (*float*) – the ratio of test set in all data.

Return (train_set, dev_set) train_set: the training set dev_set: the development set

fastNLP.core.dataset.**construct_dataset**(*sentences*)
Construct a data set from a list of sentences.

Parameters **sentences** – list of list of str

Return dataset a DataSet object

fastNLP.core.fieldarray

class fastNLP.core.fieldarray.**AutoPadder**(*pad_val=0*)
contentspadding (1) (fieldList, FieldArray.dtype[‘This’, ‘is’, . . .]

np.str, [[1,2], . . .]np.int64)(np.int64, np.float64)padding

2. (np.int64, np.float64), (2.1) fieldsequence_length, padding (2.2) fieldList, BatchList padListListpadding-padder

instancefield[1, 2, 3]pad [[1,2], [3,4, . . .]]pad

class fastNLP.core.fieldarray.**EngChar2DPadder**(*pad_val=0*, *pad_length=0*)

character2D paddingfield[[‘T’, ‘h’, ‘i’, ‘s’], [‘a’], [‘d’, ‘e’, ‘m’, ‘o’]](strcharacterindex)

paddedbatch(batch_size, max_sentence_length, max_word_length). max_sentence_length
max_word_lengthword

class fastNLP.core.fieldarray.**FieldArray**(*name*, *content*, *is_target=None*, *is_input=None*, *padder=<fastNLP.core.fieldarray.AutoPadder object>*)
FieldArray is the collection of Instance``s of the same field. It is the basic element of ``DataSet class.

Parameters

- **name** (*str*) – the name of the FieldArray
- **content** (*list*) – a list of int, float, str or np.ndarray, or a list of list of one, or a np.ndarray.
- **is_target** (*bool*) – If True, this FieldArray is used to compute loss.
- **is_input** (*bool*) – If True, this FieldArray is used to the model input.
- **padder** – PadderBasepadding(characterpadding)

append(*val*)
Add a new item to the tail of FieldArray.

Parameters **val** – int, float, str, or a list of one.

get(*indices*, *pad=True*)
Fetch instances based on indices.

Parameters

- **indices** – an int, or a list of int.
- **pad** – bool, padding

Returns

set_pad_val(*pad_val*)
padderpad_val. :param pad_val: int :return:

set_padder(*padder*)
padding

Parameters **padder** – PadderBaseNone. Nonepadder.

Returns

class fastNLP.core.fieldarray.**PadderBase**(*pad_val=0*, ***kwargs*)
padder__call__() batchpaddingelementinplaceelementinplacedeepcopy

fastNLP.core.instance

class fastNLP.core.instance.**Instance**(****fields*)
An Instance is an example of data. Example:

```
ins = Instance(field_1=[1, 1, 1], field_2=[2, 2, 2])
ins["field_1"]
>>[1, 1, 1]
ins.add_field("field_3", [3, 3, 3])
```

Parameters **fields** – a dict of (str: list).

add_field(*field_name*, *field*)
Add a new field to the instance.

Parameters **field_name** – str, the name of the field.

fastNLP.core.losses

class fastNLP.core.losses.**BCELoss**(*pred=None*, *target=None*)

class fastNLP.core.losses.**CrossEntropyLoss**(*pred=None*, *target=None*, *padding_idx=-100*)

class fastNLP.core.losses.**L1Loss**(*pred=None*, *target=None*)

class fastNLP.core.losses.**LossBase**
Base class for all losses.

class fastNLP.core.losses.**LossFunc**(*func*, *key_map=None*, ***kwargs*)
A wrapper of user-provided loss function.

class fastNLP.core.losses.**LossInForward**(*loss_key='loss'*)

class fastNLP.core.losses.**NLLLoss**(*pred=None*, *target=None*)

fastNLP.core.losses.**make_mask**(*lens*, *tar_len*)
To generate a mask over a sequence.

Parameters

- **lens** – list or LongTensor, [batch_size]
- **tar_len** – int

Return mask ByteTensor

fastNLP.core.losses.**mask**(*predict*, *truth*, ***kwargs*)
To select specific elements from Tensor. This method calls `squash()`.

Parameters

- **predict** – Tensor, [batch_size , max_len , tag_size]
- **truth** – Tensor, [batch_size , max_len]

- ****kwargs** – extra arguments, kwargs["mask"]: ByteTensor, [batch_size , max_len], the mask Tensor. The position that is 1 will be selected.

Return predict , truth predict & truth after processing

fastNLP.core.losses.**squash**(*predict*, *truth*, ***kwargs*)
To reshape tensors in order to fit loss functions in PyTorch.

Parameters

- **predict** – Tensor, model output
- **truth** – Tensor, truth from dataset
- ****kwargs** – extra arguments

Return predict , truth predict & truth after processing

fastNLP.core.losses.**unpad**(*predict*, *truth*, ***kwargs*)
To process padded sequence output to get true loss.

Parameters

- **predict** – Tensor, [batch_size , max_len , tag_size]
- **truth** – Tensor, [batch_size , max_len]
- **kwargs** – kwargs["lens"] is a list or LongTensor, with size [batch_size]. The i-th element is true lengths of i-th sequence.

Return predict , truth predict & truth after processing

fastNLP.core.losses.**unpad_mask**(*predict*, *truth*, ***kwargs*)
To process padded sequence output to get true loss.

Parameters

- **predict** – Tensor, [batch_size , max_len , tag_size]
- **truth** – Tensor, [batch_size , max_len]
- **kwargs** – kwargs["lens"] is a list or LongTensor, with size [batch_size]. The i-th element is true lengths of i-th sequence.

Return predict , truth predict & truth after processing

fastNLP.core.metrics

class fastNLP.core.metrics.**AccuracyMetric**(*pred=None*, *target=None*, *seq_lens=None*)
Accuracy Metric

evaluate(*pred*, *target*, *seq_lens=None*)

Parameters

- **pred** – List of (torch.Tensor, or numpy.ndarray). Element's shape can be: torch.Size([B,]), torch.Size([B, n_classes]), torch.Size([B, max_len]), torch.Size([B, max_len, n_classes])
- **target** – List of (torch.Tensor, or numpy.ndarray). Element's can be: torch.Size([B,]), torch.Size([B,]), torch.Size([B, max_len]), torch.Size([B, max_len])
- **seq_lens** – List of (torch.Tensor, or numpy.ndarray). Element's can be: None, None, torch.Size([B], torch.Size([B]). ignored if masks are provided.

get_metric(*reset=True*)
Returns computed metric.

Parameters **reset** (*bool*) – whether to recount next time.

Return evaluate_result {"acc": float}

class fastNLP.core.metrics.**BMESF1PreRecMetric**(*b_idx=0*, *m_idx=1*, *e_idx=2*, *s_idx=3*, *pred=None*, *target=None*, *seq_lens=None*)

BMESf1, precision, recalltag"BS"cur_BtagB next_BtagBcur_B=SpredictBtagSnext_M=B, predictMtagB | | next_B | next_M | next_E | next_S | end | |:——:|:——-:|:——:|:——-:|:——:|:——-:| | start | | next_M=B | next_E=S | | - | | cur_B | cur_B=S | | | cur_B=S | cur_B=S | | cur_M | cur_M=E | | | cur_M=E | cur_M=E | | cur_E | | next_M=B | next_E=S | | | | cur_S | | next_M=B | next_E=S | | |

predictionBSEMSSSSSS.

Metrictargettarget pred(batch_size, max_len) (batch_size, max_len, 4) target (batch_size, max_len) seq_lens (batch_size,)

class fastNLP.core.metrics.**MetricBase**
Base class for all metrics.

`MetricBase` handles validity check of its input dictionaries - `pred_dict` and `target_dict`. `pred_dict` is the output of `forward()` or prediction function of a model. `target_dict` is the ground truth from DataSet where `is_target` is set `True`. `MetricBase` will do the following type checks:

1. whether self.evaluate has varargs, which is not supported.
2. whether params needed by self.evaluate is not included in `pred_dict`, `target_dict`.
3. whether params needed by self.evaluate duplicate in `pred_dict`, `target_dict`.
4. whether params in `pred_dict`, `target_dict` are not used by evaluate.(Might cause warning)

Besides, before passing params into self.evaluate, this function will filter out params from output_dict and target_dict which are not used in self.evaluate. (but if **kwargs presented in self.evaluate, no filtering will be conducted.) However, in some cases where type check is not necessary, `_fast_param_map` will be used.

class fastNLP.core.metrics.**SpanFPreRecMetric**(*tag_vocab*, *pred=None*, *target=None*, *seq_lens=None*, *encoding_type='bio'*, *ignore_labels=None*, *only_gross=True*, *f_type='micro'*, *beta=1*)

spanF, pre, rec. metric {

'f': xxx, # ff_beta 'pre': xxx, 'rec':xxx

} only_gross=False, labelmetric

{ 'f': xxx, 'pre': xxx, 'rec':xxx, 'f-label': xxx, 'pre-label': xxx, 'rec-label':xxx, ...

}

evaluate(*pred*, *target*, *seq_lens*)
A lot of design idea comes from allennlp's measure :param pred: :param target: :param seq_lens: :return:

fastNLP.core.metrics.**accuracy_topk**(*y_true*, *y_prob*, *k=1*)
Compute accuracy of y_true matching top-k probable labels in y_prob.

Parameters

- **y_true** – ndarray, true label, [n_samples]
- **y_prob** – ndarray, label probabilities, [n_samples, n_classes]

- **k** – int, k in top-k

Returns acc accuracy of top-k

fastNLP.core.metrics.**bio_tag_to_spans**(*tags*, *ignore_labels=None*)

Parameters

- **tags** – List[str],
- **ignore_labels** – List[str], listlabel

Returns List[Tuple[str, List[int, int]]]. [(label[start, end])]

fastNLP.core.metrics.**bmes_tag_to_spans**(*tags*, *ignore_labels=None*)

Parameters

- **tags** – List[str],
- **ignore_labels** – List[str], listlabel

Returns List[Tuple[str, List[int, int]]]. [(label[start, end])]

fastNLP.core.metrics.**pred_topk**(*y_prob*, *k=1*)

Return top-k predicted labels and corresponding probabilities.

Parameters

- **y_prob** – ndarray, size [n_samples, n_classes], probabilities on labels
- **k** – int, k of top-k

Returns (y_pred_topk, y_prob_topk) y_pred_topk: ndarray, size [n_samples, k], predicted top-k labels y_prob_topk: ndarray, size [n_samples, k], probabilities for top-k labels

fastNLP.core.optimizer

class fastNLP.core.optimizer.**Adam**(*lr=0.001*, *weight_decay=0*, *betas=(0.9*, *0.999)*, *eps=1e-08*, *amsgrad=False*, *model_params=None*)

Parameters

- **lr** (*float*) – learning rate
- **weight_decay** (*float*) –
- **model_params** – a generator. E.g. `model.parameters()` for PyTorch models.

class fastNLP.core.optimizer.**Optimizer**(*model_params*, ***kwargs*)

Parameters

- **model_params** – a generator. E.g. `model.parameters()` for PyTorch models.
- **kwargs** – additional parameters.

class fastNLP.core.optimizer.**SGD**(*lr=0.001*, *momentum=0*, *model_params=None*)

Parameters

- **lr** (*float*) – learning rate. Default: 0.01
- **momentum** (*float*) – momentum. Default: 0
- **model_params** – a generator. E.g. `model.parameters()` for PyTorch models.

fastNLP.core.predictor

class fastNLP.core.predictor.**Predictor**(*network*)
An interface for predicting outputs based on trained models.

It does not care about evaluations of the model, which is different from Tester. This is a high-level model wrapper to be called by FastNLP. This class does not share any operations with Trainer and Tester. Currently, Predictor does not support GPU.

predict(*data*, *seq_len_field_name=None*)
Perform inference using the trained model.

Parameters

- **data** – a DataSet object.
- **seq_len_field_name** (*str*) – field name indicating sequence lengths

Returns list of batch outputs

fastNLP.core.sampler

class fastNLP.core.sampler.**BaseSampler**
The base class of all samplers.

Sub-classes must implement the __call__ method. __call__ takes a DataSet object and returns a list of int - the sampling indices.

class fastNLP.core.sampler.**BucketSampler**(*num_buckets=10*, *batch_size=32*, *seq_lens_field_name='seq_lens'*)

Parameters

- **num_buckets** (*int*) – the number of buckets to use.
- **batch_size** (*int*) – batch size per epoch.
- **seq_lens_field_name** (*str*) – the field name indicating the field about sequence length.

class fastNLP.core.sampler.**RandomSampler**
Sample data in random permutation order.

class fastNLP.core.sampler.**SequentialSampler**
Sample data in the original order.

fastNLP.core.sampler.**convert_to_torch_tensor**(*data_list*, *use_cuda*)
Convert lists into (cuda) Tensors.

Parameters

- **data_list** – 2-level lists
- **use_cuda** – bool, whether to use GPU or not

Return data_list PyTorch Tensor of shape [batch_size, max_seq_len]

fastNLP.core.sampler.**k_means_1d**(*x*, *k*, *max_iter=100*)
Perform k-means on 1-D data.

Parameters

- **x** – list of int, representing points in 1-D.
- **k** – the number of clusters required.

- **max_iter** – maximum iteration

Return centroids numpy array, centroids of the k clusters assignment: numpy array, 1-D, the bucket id assigned to each example.

fastNLP.core.sampler.**k_means_bucketing**(*lengths*, *buckets*)

Assign all instances into possible buckets using k-means, such that instances in the same bucket have similar lengths.

Parameters

- **lengths** – list of int, the length of all samples.
- **buckets** – list of int. The length of the list is the number of buckets. Each integer is the maximum length threshold for each bucket (This is usually None.).

Return data 2-level list

```
[
    [index_11, index_12, ...],  # bucket 1
    [index_21, index_22, ...],  # bucket 2
    ...
]
```

fastNLP.core.sampler.**simple_sort_bucketing**(*lengths*)

Parameters **lengths** – list of int, the lengths of all examples.

Return data 2-level list

```
[
    [index_11, index_12, ...],  # bucket 1
    [index_21, index_22, ...],  # bucket 2
    ...
]
```

fastNLP.core.tester

class fastNLP.core.tester.**Tester**(*data*, *model*, *metrics*, *batch_size=16*, *use_cuda=False*, *verbose=1*)

An collection of model inference and evaluation of performance, used over validation/dev set and test set.

Parameters

- **data** (DataSet) – a validation/development set
- **model** (*torch.nn.modules.module*) – a PyTorch model
- **metrics** (MetricBase) – a metric object or a list of metrics (List[MetricBase])
- **batch_size** (*int*) – batch size for validation
- **use_cuda** (*bool*) – whether to use CUDA in validation.
- **verbose** (*int*) – the number of steps after which an information is printed.

test()

Start test or validation.

Return eval_results a dictionary whose keys are the class name of metrics to use, values are the evaluation results of these metrics.

fastNLP.core.trainer

fastNLP.core.utils

exception fastNLP.core.utils.**CheckError**(*check_res: fastNLP.core.utils.CheckRes*, *func_signature: str*)

CheckError. Used in losses.LossBase, metrics.MetricBase.

class fastNLP.core.utils.**CheckRes**(*missing*, *unused*, *duplicated*, *required*, *all_needed*, *varargs*)

all_needed
Alias for field number 4

duplicated
Alias for field number 2

missing
Alias for field number 0

required
Alias for field number 3

unused
Alias for field number 1

varargs
Alias for field number 5

fastNLP.core.utils.**get_func_signature**(*func*)

Given a function or method, return its signature. For example: (1) function

def func(a, b='a', *args): xxxx

get_func_signature(func) # 'func(a, b='a', *args)'

2. method class Demo:

def __init__(self): xxx

def forward(self, a, b='a', **args)

demo = Demo() get_func_signature(demo.forward) # 'Demo.forward(self, a, b='a', **args)'

Parameters **func** – a function or a method

Returns str or None

fastNLP.core.utils.**load_pickle**(*pickle_path*, *file_name*)

Load an object from a given pickle file.

Parameters

- **pickle_path** – str, the directory where the pickle file is.
- **file_name** – str, the name of the pickle file.

Return obj an object stored in the pickle

fastNLP.core.utils.**pickle_exist**(*pickle_path*, *pickle_name*)

Check if a given pickle file exists in the directory.

Parameters

- **pickle_path** – the directory of target pickle file
- **pickle_name** – the filename of target pickle file

Returns True if file exists else False

class fastNLP.core.utils.**pseudo_tqdm**(***kwargs*)
tqdmTraineruse_tqdmfalse

fastNLP.core.utils.**save_pickle**(*obj*, *pickle_path*, *file_name*)
Save an object into a pickle file.

Parameters

- **obj** – an object
- **pickle_path** – str, the directory where the pickle file is to be saved
- **file_name** – str, the name of the pickle file. In general, it should be ended by "pkl".

fastNLP.core.utils.**seq_lens_to_masks**(*seq_lens*, *float=False*)
Convert seq_lens to masks. :param seq_lens: list, np.ndarray, or torch.LongTensor, shape should all be (B,) :param float: if True, the return masks is in float type, otherwise it is byte. :return: list, np.ndarray or torch.Tensor, shape will be (B, max_length)

fastNLP.core.utils.**seq_mask**(*seq_len*, *max_len*)
Create sequence mask.

Parameters

- **seq_len** – list or torch.Tensor, the lengths of sequences in a batch.
- **max_len** – int, the maximum sequence length in a batch.

Return mask torch.LongTensor, [batch_size, max_len]

fastNLP.core.vocabulary

class fastNLP.core.vocabulary.**Vocabulary**(*max_size=None*, *min_freq=None*, *unknown='<unk>'*, *padding='<pad>'*)
Use for word and index one to one mapping

Example:

```
vocab = Vocabulary()
word_list = "this is a word list".split()
vocab.update(word_list)
vocab["word"]
vocab.to_word(5)
```

Parameters

- **max_size** (*int*) – set the max number of words in Vocabulary. Default: None
- **min_freq** (*int*) – set the min occur frequency of words in Vocabulary. Default: None

build_reverse_vocab()
Build "index to word" dict based on "word to index" dict.

build_vocab()
Build a mapping from word to index, and filter the word using max_size and min_freq.

to_index(*w*)
Turn a word to an index. If w is not in Vocabulary, return the unknown label.

Parameters **w** (*str*) – a word

fastNLP.core.vocabulary.**check_build_status**(*func*)
A decorator to check whether the vocabulary updates after the last build.

fastNLP.core.vocabulary.**check_build_vocab**(*func*)
A decorator to make sure the indexing is built before used.

3.1.3 fastNLP.io

fastNLP.io.base_loader

class fastNLP.io.base_loader.**BaseLoader**
Base loader for all loaders.

classmethod load(*data_path*)
list of list of str

static load_lines(*data_path*)
list of str

classmethod load_with_cache(*data_path*, *cache_path*)
load

class fastNLP.io.base_loader.**DataLoaderRegister**
Register for all data sets.

fastNLP.io.config_io

class fastNLP.io.config_io.**ConfigLoader**(*data_path=None*)
Loader for configuration.

Parameters **data_path** (*str*) – path to the config

static load_config(*file_path*, *sections*)
Load section(s) of configuration into the `sections` provided. No returns.

Parameters

- **file_path** (*str*) – the path of config file
- **sections** (*dict*) – the dict of `{section_name(string): ConfigSection object}`

Example:

```
test_args = ConfigSection()
ConfigLoader("config.cfg").load_config("./data_for_tests/config", {"POS_test
↪": test_args})
```

class fastNLP.io.config_io.**ConfigSaver**(*file_path*)
ConfigSaver is used to save config file and solve related conflicts.

Parameters **file_path** (*str*) – path to the config file

save_config_file(*section_name*, *section*)
This is the function to be called to change the config file with a single section and its name.

Parameters

- **section_name** (*str*) – The name of section what needs to be changed and saved.
- **section** (ConfigSection) – The section with key and value what needs to be changed and saved.

class fastNLP.io.config_io.**ConfigSection**
ConfigSection is the data structure storing all key-value pairs in one section in a config file.

fastNLP.io.dataset_loader

class fastNLP.io.dataset_loader.**Conll2003Loader**
Loader for conll2003 dataset

More information about the given dataset cound be found on https://sites.google.com/site/ermasoftware/getting-started/ne-tagging-conll2003-data

convert(*parsed_data*)
Optional operation to build a DataSet.

Parameters **data** – inner data structure (user-defined) to represent the data.

Returns a DataSet object

load(*dataset_path*)
Load data from a given file.

Parameters **path** (*str*) – file path

Returns a DataSet object

class fastNLP.io.dataset_loader.**ConllLoader**
loader for conll format files

convert(*data*)
Optional operation to build a DataSet.

Parameters **data** – inner data structure (user-defined) to represent the data.

Returns a DataSet object

load(*data_path*)
Load data from a given file.

Parameters **path** (*str*) – file path

Returns a DataSet object

static parse(*lines*)

Parameters **lines** (*list*) – a list containing all lines in a conll file.

Returns a 3D list

class fastNLP.io.dataset_loader.**ConllxDataLoader**
“”“ZhConllPOSReader‘‘

class fastNLP.io.dataset_loader.**DataSetLoader**
Interface for all DataSetLoaders.

convert(*data*)
Optional operation to build a DataSet.

Parameters **data** – inner data structure (user-defined) to represent the data.

Returns a DataSet object

load(*path*)
Load data from a given file.

Parameters **path** (*str*) – file path

Returns a DataSet object

class fastNLP.io.dataset_loader.**DummyCWSReader**
Load pku dataset for Chinese word segmentation.

convert(*data*)
Optional operation to build a DataSet.

Parameters **data** – inner data structure (user-defined) to represent the data.

Returns a DataSet object

load(*data_path*, *max_seq_len=32*)
Load pku dataset for Chinese word segmentation. CWS (Chinese Word Segmentation) pku training dataset format: 1. Each line is a sentence. 2. Each word in a sentence is separated by space. This function convert the pku dataset into three-level lists with labels <BMES>. B: beginning of a word M: middle of a word E: ending of a word S: single character

Parameters

- **data_path** (*str*) – path to the data set.
- **max_seq_len** – int, the maximum length of a sequence. If a sequence is longer than it, split it into several sequences.

Returns three-level lists

class fastNLP.io.dataset_loader.**DummyClassificationReader**
Loader for a dummy classification data set

convert(*data*)
Optional operation to build a DataSet.

Parameters **data** – inner data structure (user-defined) to represent the data.

Returns a DataSet object

load(*data_path*)
Load data from a given file.

Parameters **path** (*str*) – file path

Returns a DataSet object

static parse(*lines*)
token/

Parameters **lines** – lines from dataset

Returns list(list(list())): the three level of lists are words, sentence, and dataset

class fastNLP.io.dataset_loader.**DummyLMReader**
A Dummy Language Model Dataset Reader

convert(*data*)
Optional operation to build a DataSet.

Parameters **data** – inner data structure (user-defined) to represent the data.

Returns a DataSet object

load(*data_path*)
Load data from a given file.

Parameters **path** (*str*) – file path

Returns a DataSet object

class fastNLP.io.dataset_loader.**DummyPOSReader**
A simple reader for a dummy POS tagging dataset.

In these datasets, each line are divided by " ". The first Col is the vocabulary and the second Col is the label. Different sentence are divided by an empty line.

E.g:

```
Tom label1
and label2
Jerry   label1
.   label3
(separated by an empty line)
Hello   label4
world   label5
!   label3
```

In this example, there are two sentences "Tom and Jerry ." and "Hello world !". Each word has its own label.

convert(*data*)
Convert lists of strings into Instances with Fields.

load(*data_path*)

Return data three-level list Example:

```
[
    [ [word_11, word_12, ...], [label_1, label_1, ...] ],
    [ [word_21, word_22, ...], [label_2, label_1, ...] ],
    ...
]
```

class fastNLP.io.dataset_loader.**NaiveCWSReader**(*in_word_splitter=None*)
reader, :

```
 fastNLP ,   good   .
```

,partpos tag :

```
/D  /P  /Na  /Ng  /COMMACATEGORY
```

load(*filepath*, *in_word_splitter=None*, *cut_long_sent=False*)

(seg) fastNLP , good .

/D /P /Na /Ng /COMMACATEGORY

splitterNone, splitter"/D", . "/D".split('/')[0]

Parameters

- **filepath** –
- **in_word_splitter** –
- **cut_long_sent** –

Returns

class fastNLP.io.dataset_loader.**NativeDataSetLoader**

A simple example of DataSetLoader

load(*path*)

Load data from a given file.

Parameters **path** (*str*) – file path

Returns a DataSet object

class fastNLP.io.dataset_loader.**PeopleDailyCorpusLoader** **convert**(*data*)

Optional operation to build a DataSet.

Parameters **data** – inner data structure (user-defined) to represent the data.

Returns a DataSet object

load(*data_path*, *pos=True*, *ner=True*)

Parameters

- **data_path** (*str*) –
- **pos** (*bool*) –
- **ner** (*bool*) –

Returns a DataSet object

class fastNLP.io.dataset_loader.**RawDataSetLoader**

A simple example of raw data reader

convert(*data*)

Optional operation to build a DataSet.

Parameters **data** – inner data structure (user-defined) to represent the data.

Returns a DataSet object

load(*data_path*, *split=None*)

Load data from a given file.

Parameters **path** (*str*) – file path

Returns a DataSet object

class fastNLP.io.dataset_loader.**SNLIDataSetReader**

A data set loader for SNLI data set.

convert(*data*)

Convert a 3D list to a DataSet object.

Parameters **data** – A 3D tensor. Example:

```
[
    [ [premise_word_11, premise_word_12, ...], [hypothesis_word_11,␣
↪hypothesis_word_12, ...], [label_1] ],
    [ [premise_word_21, premise_word_22, ...], [hypothesis_word_21,␣
↪hypothesis_word_22, ...], [label_2] ],
    ...
]
```

Returns A DataSet object.

load(*path_list*)

Parameters **path_list** (*list*) – A list of file name, in the order of premise file, hypothesis file, and label file.

Returns A DataSet object.

class fastNLP.io.dataset_loader.**ZhConllPOSReader**

Conll“”BMES

load(*path*)

DataSet, field wordslist of str, tag: list of str, BMES tag, [‘VP’, ‘NN’, ‘NN’, ..][“S-VP”, “B-NN”, “M-NN”,..]

conll7

```
1              NN      O       11      nmod:topic
2                  PU      O       11      punct
3   7      7       NT      DATE    4       compound:nn
4   12     12      NT      DATE    11      nmod:tmod
5                  PU      O       11      punct

1                  DT      O       3       det
2                  M       O       1       mark:clf
3                NN      O       8       nsubj
4                  P       O       5       case
5                NN      O       8       nmod:prep
```

fastNLP.io.dataset_loader.**add_seg_tag**(*data*)

Parameters **data** – list of ([word], [pos], [heads], [head_tags])

Returns list of ([word], [pos])

fastNLP.io.dataset_loader.**convert_seq2seq_dataset**(*data*)

Convert list of data into DataSet.

Parameters **data** – list of list of strings, [num_examples, *]. Example:

```
[
    [ [word_11, word_12, ...], [label_1, label_1, ...] ],
    [ [word_21, word_22, ...], [label_2, label_1, ...] ],
    ...
]
```

Returns a DataSet.

fastNLP.io.dataset_loader.**convert_seq2tag_dataset**(*data*)

Convert list of data into DataSet.

Parameters **data** – list of list of strings, [num_examples, *]. Example:

```
[
    [ [word_11, word_12, ...], label_1 ],
    [ [word_21, word_22, ...], label_2 ],
    ...
]
```

Returns a DataSet.

fastNLP.io.dataset_loader.**convert_seq_dataset**(*data*)

Create an DataSet instance that contains no labels.

Parameters **data** – list of list of strings, [num_examples, *]. Example:

```
[
    [word_11, word_12, ...],
    ...
]
```

Returns a DataSet.

fastNLP.io.dataset_loader.**cut_long_sentence**(*sent*, *max_sample_length=200*)
max_sample_lengthsentencemax_sample_length

Parameters

- **sent** – str.
- **max_sample_length** – int.

Returns list of str.

fastNLP.io.embed_loader

class fastNLP.io.embed_loader.**EmbedLoader**
docstring for EmbedLoader

static fast_load_embedding(*emb_dim*, *emb_file*, *vocab*)
Fast load the pre-trained embedding and combine with the given dictionary. This loading method uses line-by-line operation.

Parameters

- **emb_dim** (*int*) – the dimension of the embedding. Should be the same as pre-trained embedding.
- **emb_file** (*str*) – the pre-trained embedding file path.
- **vocab** (Vocabulary) – a mapping from word to index, can be provided by user or built from pre-trained embedding

Return embedding_matrix numpy.ndarray

static load_embedding(*emb_dim*, *emb_file*, *emb_type*, *vocab*)
Load the pre-trained embedding and combine with the given dictionary.

Parameters

- **emb_dim** (*int*) – the dimension of the embedding. Should be the same as pre-trained embedding.
- **emb_file** (*str*) – the pre-trained embedding file path.
- **emb_type** (*str*) – the pre-trained embedding format, support glove now
- **vocab** (Vocabulary) – a mapping from word to index, can be provided by user or built from pre-trained embedding

Return (embedding_tensor, vocab) embedding_tensor - Tensor of shape (len(word_dict), emb_dim); vocab - input vocab or vocab built by pre-train

fastNLP.io.logger

fastNLP.io.logger.**create_logger**(*logger_name*, *log_path*, *log_format=None*, *log_level=20*)
Create a logger.

Parameters

- **logger_name** (*str*) –
- **log_path** (*str*) –
- **log_format** –
- **log_level** –

Returns logger

To use a logger:

```
logger.debug("this is a debug message")
logger.info("this is a info message")
logger.warning("this is a warning message")
logger.error("this is an error message")
```

fastNLP.io.model_io

class fastNLP.io.model_io.**ModelLoader**

Loader for models.

static load_pytorch(*empty_model*, *model_path*)

Load model parameters from ".pkl" files into the empty PyTorch model.

Parameters

- **empty_model** – a PyTorch model with initialized parameters.
- **model_path** (*str*) – the path to the saved model.

static load_pytorch_model(*model_path*)

Load the entire model.

Parameters **model_path** (*str*) – the path to the saved model.

class fastNLP.io.model_io.**ModelSaver**(*save_path*)

Save a model

Parameters **save_path** (*str*) – the path to the saving directory.

Example:

```
saver = ModelSaver("./save/model_ckpt_100.pkl")
saver.save_pytorch(model)
```

save_pytorch(*model*, *param_only=True*)

Save a pytorch model into ".pkl" file.

Parameters

- **model** – a PyTorch model
- **param_only** (*bool*) – whether only to save the model parameters or the entire model.

3.1.4 fastNLP.models

fastNLP.models.base_model

class fastNLP.models.base_model.**BaseModel**
Base PyTorch model for all models.

class fastNLP.models.base_model.**NaiveClassifier**(*in_feature_dim*, *out_feature_dim*)

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.

---

fastNLP.models.biaffine_parser

class fastNLP.models.biaffine_parser.**ArcBiaffine**(*hidden_size*, *bias=True*)
helper module for Biaffine Dependency Parser predicting arc

forward(*head*, *dep*)
:param head arc-head tensor = [batch, length, emb_dim] :param dep arc-dependent tensor = [batch, length, emb_dim]

:return output tensor = [bacth, length, length]

class fastNLP.models.biaffine_parser.**BiaffineParser**(*word_vocab_size*, *word_emb_dim*, *pos_vocab_size*, *pos_emb_dim*, *num_label*, *rnn_layers=1*, *rnn_hidden_size=200*, *arc_mlp_size=100*, *label_mlp_size=100*, *dropout=0.3*, *encoder='lstm'*, *use_greedy_infer=False*)
Biaffine Dependency Parser implemantation. refer to ' Deep Biaffine Attention for Neural Dependency Parsing (Dozat and Manning, 2016) <https://arxiv.org/abs/1611.01734>'_ .

forward(*word_seq*, *pos_seq*, *seq_lens*, *gold_heads=None*)

Parameters

- **word_seq** – [batch_size, seq_len] sequence of word's indices
- **pos_seq** – [batch_size, seq_len] sequence of word's indices
- **seq_lens** – [batch_size, seq_len] sequence of length masks
- **gold_heads** – [batch_size, seq_len] sequence of golden heads

Return dict parsing results arc_pred: [batch_size, seq_len, seq_len] label_pred: [batch_size, seq_len, seq_len] mask: [batch_size, seq_len] head_pred: [batch_size, seq_len] if gold_heads is not provided, predicting the heads

static loss(*arc_pred*, *label_pred*, *arc_true*, *label_true*, *mask*)
Compute loss.

Parameters

- **arc_pred** – [batch_size, seq_len, seq_len]
- **label_pred** – [batch_size, seq_len, n_tags]
- **arc_true** – [batch_size, seq_len]
- **label_true** – [batch_size, seq_len]
- **mask** – [batch_size, seq_len]

Returns loss value

predict(*word_seq*, *pos_seq*, *seq_lens*)

Parameters

- **word_seq** –
- **pos_seq** –
- **seq_lens** –

Returns arc_pred: [B, L] label_pred: [B, L]

class fastNLP.models.biaffine_parser.**GraphParser**

Graph based Parser helper class, support greedy decoding and MST(Maximum Spanning Tree) decoding

class fastNLP.models.biaffine_parser.**LabelBilinear**(*in1_features*, *in2_features*, *num_label*, *bias=True*)

helper module for Biaffine Dependency Parser predicting label

forward(*x1*, *x2*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

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

---

class fastNLP.models.biaffine_parser.**ParserLoss**(*arc_pred=None*, *label_pred=None*, *arc_true=None*, *label_true=None*)

class fastNLP.models.biaffine_parser.**ParserMetric**(*arc_pred=None*, *label_pred=None*, *arc_true=None*, *label_true=None*, *seq_lens=None*)

evaluate(*arc_pred*, *label_pred*, *arc_true*, *label_true*, *seq_lens=None*)

Evaluate the performance of prediction.

fastNLP.models.biaffine_parser.**mst**(*scores*)

with some modification to support parser output for MST decoding https://github.com/tdozat/Parser/blob/0739216129cd39d69997d28cbc4133b360ea3934/lib/models/nn.py#L692

fastNLP.models.char_language_model

class fastNLP.models.char_language_model.**CharLM**(*char_emb_dim*, *word_emb_dim*, *vocab_size*, *num_char*)

CNN + highway network + LSTM # Input:

4D tensor with shape [batch_size, in_channel, height, width]

 # Output: 2D Tensor with shape [batch_size, vocab_size]

 # Arguments: char_emb_dim: the size of each character's attention word_emb_dim: the size of each word's attention vocab_size: num of unique words num_char: num of characters use_gpu: True or False

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

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

---

class fastNLP.models.char_language_model.**Highway**(*input_size*)

Highway network

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.

---

fastNLP.models.cnn_text_classification

class fastNLP.models.cnn_text_classification.**CNNText**(*embed_num*, *embed_dim*, *num_classes*, *kernel_nums=(3, 4, 5)*, *kernel_sizes=(3, 4, 5)*, *padding=0*, *dropout=0.5*)

Text classification model by character CNN, the implementation of paper 'Yoon Kim. 2014. Convolution Neural Networks for Sentence Classification.'

forward(*word_seq*)

Parameters **word_seq** – torch.LongTensor, [batch_size, seq_len]

Return output dict of torch.LongTensor, [batch_size, num_classes]

predict(*word_seq*)

Parameters **word_seq** – torch.LongTensor, [batch_size, seq_len]

Return predict dict of torch.LongTensor, [batch_size, seq_len]

fastNLP.models.sequence_modeling

class fastNLP.models.sequence_modeling.**AdvSeqLabel**(*args*, *emb=None*, *id2words=None*)

Advanced Sequence Labeling Model

forward(*word_seq*, *word_seq_origin_len*, *truth=None*)

Parameters

- **word_seq** – LongTensor, [batch_size, mex_len]
- **word_seq_origin_len** – LongTensor, [batch_size,]
- **truth** – LongTensor, [batch_size, max_len]

Return y If truth is None, return list of [decode path(list)]. Used in testing and predicting. If truth is not None, return loss, a scalar. Used in training.

loss(***kwargs*)
Since the loss has been computed in forward(), this function simply returns x.

class fastNLP.models.sequence_modeling.**SeqLabeling**(*args*)
PyTorch Network for sequence labeling

decode(*x*, *pad=True*)

Parameters

- **x** – FloatTensor, [batch_size, max_len, tag_size]
- **pad** – pad the output sequence to equal lengths

Return prediction list of [decode path(list)]

forward(*word_seq*, *word_seq_origin_len*, *truth=None*)

Parameters

- **word_seq** – LongTensor, [batch_size, mex_len]
- **word_seq_origin_len** – LongTensor, [batch_size,], the origin lengths of the sequences.
- **truth** – LongTensor, [batch_size, max_len]

Return y If truth is None, return list of [decode path(list)]. Used in testing and predicting. If truth is not None, return loss, a scalar. Used in training.

loss(*x*, *y*)
Since the loss has been computed in forward(), this function simply returns x.

fastNLP.models.snli

class fastNLP.models.snli.**ESIM**(***kwargs*)
PyTorch Network for SNLI task using ESIM model.

forward(*premise*, *hypothesis*, *premise_len*, *hypothesis_len*)
Forward function

Parameters

- **premise** – A Tensor represents premise: [batch size(B), premise seq len(PL)].
- **hypothesis** – A Tensor represents hypothesis: [B, hypothesis seq len(HL)].
- **premise_len** – A Tensor record which is a real word and which is a padding word in premise: [B, PL].
- **hypothesis_len** – A Tensor record which is a real word and which is a padding word in hypothesis: [B, HL].

Returns prediction: A Dict with Tensor of classification result: [B, n_labels(N)].

3.1.5 fastNLP.modules

fastNLP.modules.aggregator

fastNLP.modules.aggregator.attention

class fastNLP.modules.aggregator.attention.**Attention**(*normalize=False*)

forward(*query*, *memory*, *mask*)
Defines the computation performed at every call.

Should be overridden by all subclasses.

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

---

class fastNLP.modules.aggregator.attention.**Bi_Attention**

forward(*in_x1*, *in_x2*, *x1_len*, *x2_len*)
Defines the computation performed at every call.

Should be overridden by all subclasses.

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

---

class fastNLP.modules.aggregator.attention.**DotAtte**(*key_size*, *value_size*, *dropout=0.1*)

forward(*Q*, *K*, *V*, *mask_out=None*)

Parameters

- **Q** – [batch, seq_len, key_size]
- **K** – [batch, seq_len, key_size]
- **V** – [batch, seq_len, value_size]
- **mask_out** – [batch, seq_len]

class fastNLP.modules.aggregator.attention.**MultiHeadAtte**(*input_size*, *key_size*, *value_size*, *num_head*, *dropout=0.1*)

forward(*Q*, *K*, *V*, *atte_mask_out=None*)

Parameters

- **Q** – [batch, seq_len, model_size]
- **K** – [batch, seq_len, model_size]
- **V** – [batch, seq_len, model_size]

- **seq_mask** – [batch, seq_len]

fastNLP.modules.aggregator.avg_pool

class fastNLP.modules.aggregator.avg_pool.**AvgPool**(*stride=None*, *padding=0*)
1-d average pooling module.

forward(*x*)
Defines the computation performed at every call.

Should be overridden by all subclasses.

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

---

class fastNLP.modules.aggregator.avg_pool.**MeanPoolWithMask**

forward(*tensor*, *mask*, *dim=0*)
Defines the computation performed at every call.

Should be overridden by all subclasses.

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

---

fastNLP.modules.aggregator.kmax_pool

class fastNLP.modules.aggregator.kmax_pool.**KMaxPool**(*k=1*)
K max-pooling module.

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.

---

fastNLP.modules.aggregator.max_pool

class fastNLP.modules.aggregator.max_pool.**MaxPool**(*stride=None*, *padding=0*, *dilation=1*)
1-d max-pooling module.

forward(*x*)
Defines the computation performed at every call.

Should be overridden by all subclasses.

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

---

class fastNLP.modules.aggregator.max_pool.**MaxPoolWithMask**

forward(*tensor*, *mask*, *dim=0*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

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

---

fastNLP.modules.aggregator.self_attention

class fastNLP.modules.aggregator.self_attention.**SelfAttention**(*input_size*, *attention_unit=350*, *attention_hops=10*, *drop=0.5*, *initial_method=None*, *use_cuda=False*)

Self Attention Module.

Parameters

- **input_size** (*int*) –
- **attention_unit** (*int*) –
- **attention_hops** (*int*) –
- **drop** (*float*) –
- **initial_method** (*str*) –
- **use_cuda** (*bool*) –

forward(*input*, *input_origin*)

Parameters

- **input** – the matrix to do attention. [baz, senLen, h_dim]
- **inp** – then token index include pad token(0) [baz , senLen]

Return output1 the input matrix after attention operation [baz, multi-head , h_dim]

Return output2 the attention penalty term, a scalar [1]

penalization(*attention*)

compute the penalization term for attention module

fastNLP.modules.decoder

fastNLP.modules.decoder.CRF

class fastNLP.modules.decoder.CRF.**ConditionalRandomField**(*num_tags*, *include_start_end_trans=False*, *allowed_transitions=None*, *initial_method=None*)

Parameters

- **num_tags** (*int*) –
- **include_start_end_trans** (*bool*) – tag
- **allowed_transitions** (*list*) – List[Tuple[from_tag_id(int), to_tag_id(int)]]. allowed_transitions() None
- **initial_method** (*str*) –

forward(*feats*, *tags*, *mask*)

Calculate the neg log likelihood :param feats:FloatTensor, batch_size x max_len x num_tags :param tags:LongTensor, batch_size x max_len :param mask:ByteTensor batch_size x max_len :return:FloatTensor, batch_size

viterbi_decode(*data*, *mask*, *get_score=False*, *unpad=False*)

Given a feats matrix, return best decode path and best score.

:param data:FloatTensor, batch_size x max_len x num_tags :param mask:ByteTensor batch_size x max_len :param get_score: bool, whether to output the decode score. :param unpad: bool, unpad,

False, batch_size x max_lentensor TrueList[List[int]], List[int]sequencelabelunpadding

List[int]sample

Returns

get_scoreFalseunpadding get_scoreTrue, (paths, List[float],)(unpad)List[Float]

seqence

fastNLP.modules.decoder.CRF.**allowed_transitions**(*id2label*, *encoding_type='bio'*)

Parameters

- **id2label** (*dict*) – keylabelindicesvaluestrtagtag-labelvaluetag, "B", "M"; "B-NN", "M-NN", taglabel"-"Vocabulary.get_id2word()id2label
- **encoding_type** – str, "bio", "bmes"

Returns List[Tuple(int, int)]], Tuple(from_tag_id, to_tag_id) startend"BIO"BO I(start_idx, B_idx), (start_idx, O_idx), (start_idx, I_idx). start_idx=len(id2label), end_idx=len(id2label)+1

fastNLP.modules.decoder.CRF.**is_transition_allowed**(*encoding_type*, *from_tag*, *from_label*, *to_tag*, *to_label*)

Parameters

- **encoding_type** – str, "BIO", "BMES"
- **from_tag** – str, "B", "M"tag. start, endtag
- **from_label** – str, "PER", "LOC"label

- **to_tag** – str, ”B”, “M”tag. start, endtag
- **to_label** – str, ”PER”, “LOC”label

Returns bool

fastNLP.modules.decoder.MLP

class fastNLP.modules.decoder.MLP.**MLP**(*size_layer*, *activation='relu'*, *initial_method=None*, *dropout=0.0*)

Multilayer Perceptrons as a decoder

Parameters

- **size_layer** (*list*) – list of int, define the size of MLP layers.
- **activation** (*str*) – str or function, the activation function for hidden layers.
- **initial_method** (*str*) – the name of initialization method.
- **dropout** (*float*) – the probability of dropout.

Note: There is no activation function applying on output layer.

---

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.

---

fastNLP.modules.encoder

fastNLP.modules.encoder.char_embedding

class fastNLP.modules.encoder.char_embedding.**ConvCharEmbedding**(*char_emb_size=50*, *feature_maps=(40, 30, 30)*, *kernels=(3, 4, 5)*, *initial_method=None*)

Character-level Embedding with CNN.

Parameters

- **char_emb_size** (*int*) – the size of character level embedding. Default: 50 say 26 characters, each embedded to 50 dim vector, then the input_size is 50.
- **feature_maps** (*tuple*) – tuple of int. The length of the tuple is the number of convolution operations over characters. The i-th integer is the number of filters (dim of out channels) for the i-th convolution.
- **kernels** (*tuple*) – tuple of int. The width of each kernel.

forward(*x*)

Parameters **x** – [batch_size * sent_length, word_length, char_emb_size]

Returns feature map of shape [batch_size * sent_length, sum(feature_maps), 1]

class fastNLP.modules.encoder.char_embedding.**LSTMCharEmbedding**(*char_emb_size=50*, *hidden_size=None*, *initial_method=None*)

Character-level Embedding with LSTM.

Parameters

- **char_emb_size** (*int*) – the size of character level embedding. Default: 50 say 26 characters, each embedded to 50 dim vector, then the input_size is 50.
- **hidden_size** (*int*) – the number of hidden units. Default: equal to char_emb_size.

forward(*x*)

Parameters **x** – [n_batch*n_word, word_length, char_emb_size]

Returns [n_batch*n_word, char_emb_size]

fastNLP.modules.encoder.conv

class fastNLP.modules.encoder.conv.**Conv**(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*, *activation='relu'*, *initial_method=None*)

Basic 1-d convolution module, initialized with xavier_uniform.

Parameters

- **in_channels** (*int*) –
- **out_channels** (*int*) –
- **kernel_size** (*tuple*) –
- **stride** (*int*) –
- **padding** (*int*) –
- **dilation** (*int*) –
- **groups** (*int*) –
- **bias** (*bool*) –
- **activation** (*str*) –
- **initial_method** (*str*) –

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.

fastNLP.modules.encoder.conv_maxpool

class fastNLP.modules.encoder.conv_maxpool.**ConvMaxpool**(*in_channels*, *out_channels*, *kernel_sizes*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*, *activation='relu'*, *initial_method=None*)

Convolution and max-pooling module with multiple kernel sizes.

Parameters

- **in_channels** (*int*) –
- **out_channels** (*int*) –
- **kernel_sizes** (*tuple*) –
- **stride** (*int*) –
- **padding** (*int*) –
- **dilation** (*int*) –
- **groups** (*int*) –
- **bias** (*bool*) –
- **activation** (*str*) –
- **initial_method** (*str*) –

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.

fastNLP.modules.encoder.embedding

class fastNLP.modules.encoder.embedding.**Embedding**(*nums*, *dims*, *padding_idx=0*, *sparse=False*, *init_emb=None*, *dropout=0.0*)

A simple lookup table.

Parameters

- **nums** (*int*) – the size of the lookup table
- **dims** (*int*) – the size of each vector

- **padding_idx** (*int*) – pads the tensor with zeros whenever it encounters this index
- **sparse** (*bool*) – If True, gradient matrix will be a sparse tensor. In this case, only optim.SGD(cuda and cpu) and optim.Adagrad(cpu) can be used

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.

---

fastNLP.modules.encoder.linear

class fastNLP.modules.encoder.linear.**Linear**(*input_size*, *output_size*, *bias=True*, *initial_method=None*)

Parameters

- **input_size** (*int*) – input size
- **output_size** (*int*) – output size
- **bias** (*bool*) –
- **initial_method** (*str*) –

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.

---

fastNLP.modules.encoder.lstm

class fastNLP.modules.encoder.lstm.**LSTM**(*input_size*, *hidden_size=100*, *num_layers=1*, *dropout=0.0*, *batch_first=True*, *bidirectional=False*, *bias=True*, *initial_method=None*, *get_hidden=False*)

Long Short Term Memory

Parameters

- **input_size** (*int*) –
- **hidden_size** (*int*) –
- **num_layers** (*int*) –
- **dropout** (*float*) –
- **batch_first** (*bool*) –

- **bidirectional** (*bool*) –
- **bias** (*bool*) –
- **initial_method** (*str*) –
- **get_hidden** (*bool*) –

forward(*x*, *h0=None*, *c0=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.

---

fastNLP.modules.encoder.masked_rnn

class fastNLP.modules.encoder.masked_rnn.**MaskedGRU**(**args*, ***kwargs*)

Applies a multi-layer gated recurrent unit (GRU) RNN to an input sequence. For each element in the input sequence, each layer computes the following function:

$$
\begin{aligned}
r_t &= \text{sigmoid}(W_{ir}x_t + b_{ir} + W_{hr}h_{(t-1)} + b_{hr}) \\
z_t &= \text{sigmoid}(W_{iz}x_t + b_{iz} + W_{hz}h_{(t-1)} + b_{hz}) \\
n_t &= \tanh(W_{in}x_t + b_{in} + r_t * (W_{hn}h_{(t-1)} + b_{hn})) \\
h_t &= (1 - z_t) * n_t + z_t * h_{(t-1)}
\end{aligned}
$$

where h_t is the hidden state at time *t*, x_t is the hidden state of the previous layer at time *t* or $input_t$ for the first layer, and r_t, z_t, n_t are the reset, input, and new gates, respectively.

Parameters

- **input_size** (*int*) – The number of expected features in the input x
- **hidden_size** (*int*) – The number of features in the hidden state h
- **num_layers** (*int*) – Number of recurrent layers.
- **nonlinearity** (*str*) – The non-linearity to use [‘tanh’|’relu’]. Default: ‘tanh’
- **bias** (*bool*) – If False, then the layer does not use bias weights b_ih and b_hh. Default: True
- **batch_first** (*bool*) – If True, then the input and output tensors are provided as (batch, seq, feature)
- **dropout** (*bool*) – If non-zero, introduces a dropout layer on the outputs of each RNN layer except the last layer
- **bidirectional** (*bool*) – If True, becomes a bidirectional RNN. Default: False

Inputs: input, mask, h_0

- **input** (seq_len, batch, input_size): tensor containing the features of the input sequence. **mask** (seq_len, batch): 0-1 tensor containing the mask of the input sequence.
- **h_0** (num_layers * num_directions, batch, hidden_size): tensor containing the initial hidden state for each element in the batch.

Outputs: output, h_n

- **output** (seq_len, batch, hidden_size * num_directions): tensor containing the output features (h_k) from the last layer of the RNN, for each k. If a `torch.nn.utils.rnn.PackedSequence` has been given as the input, the output will also be a packed sequence.
- **h_n** (num_layers * num_directions, batch, hidden_size): tensor containing the hidden state for k=seq_len.

class fastNLP.modules.encoder.masked_rnn.**MaskedLSTM**(**args*, ***kwargs*)

Applies a multi-layer long short-term memory (LSTM) RNN to an input sequence. For each element in the input sequence, each layer computes the following function.

$$
\begin{aligned}
i_t &= \text{sigmoid}(W_{ii}x_t + b_{ii} + W_{hi}h_{(t-1)} + b_{hi}) \\
f_t &= \text{sigmoid}(W_{if}x_t + b_{if} + W_{hf}h_{(t-1)} + b_{hf}) \\
g_t &= \tanh(W_{ig}x_t + b_{ig} + W_{hc}h_{(t-1)} + b_{hg}) \\
o_t &= \text{sigmoid}(W_{io}x_t + b_{io} + W_{ho}h_{(t-1)} + b_{ho}) \\
c_t &= f_t * c_{(t-1)} + i_t * g_t \\
h_t &= o_t * \tanh(c_t)
\end{aligned}
$$

where h_t is the hidden state at time *t*, c_t is the cell state at time *t*, x_t is the hidden state of the previous layer at time *t* or $input_t$ for the first layer, and i_t, f_t, g_t, o_t are the input, forget, cell, and out gates, respectively.

Parameters

- **input_size** (*int*) – The number of expected features in the input x
- **hidden_size** (*int*) – The number of features in the hidden state h
- **num_layers** (*int*) – Number of recurrent layers.
- **bias** (*bool*) – If False, then the layer does not use bias weights b_ih and b_hh. Default: True
- **batch_first** (*bool*) – If True, then the input and output tensors are provided as (batch, seq, feature)
- **dropout** (*bool*) – If non-zero, introduces a dropout layer on the outputs of each RNN layer except the last layer
- **bidirectional** (*bool*) – If True, becomes a bidirectional RNN. Default: False

Inputs: input, mask, (h_0, c_0)

- **input** (seq_len, batch, input_size): tensor containing the features of the input sequence. **mask** (seq_len, batch): 0-1 tensor containing the mask of the input sequence.
- **h_0** (num_layers * num_directions, batch, hidden_size): tensor containing the initial hidden state for each element in the batch.
- **c_0** (num_layers * num_directions, batch, hidden_size): tensor containing the initial cell state for each element in the batch.

Outputs: output, (h_n, c_n)

- **output** (seq_len, batch, hidden_size * num_directions): tensor containing the output features *(h_t)* from the last layer of the RNN, for each t. If a `torch.nn.utils.rnn.PackedSequence` has been given as the input, the output will also be a packed sequence.
- **h_n** (num_layers * num_directions, batch, hidden_size): tensor containing the hidden state for t=seq_len
- **c_n** (num_layers * num_directions, batch, hidden_size): tensor containing the cell state for t=seq_len

class fastNLP.modules.encoder.masked_rnn.**MaskedRNN**(**args*, ***kwargs*)

Applies a multi-layer Elman RNN with costomized non-linearity to an input sequence. For each element in the input sequence, each layer computes the following function. $h_t = \tanh(w_{ih} * x_t + b_{ih} + w_{hh} * h_{(t-1)} + b_{hh})$

where h_t is the hidden state at time *t*, and x_t is the hidden state of the previous layer at time *t* or $input_t$ for the first layer. If nonlinearity='relu', then *ReLU* is used instead of *tanh*.

Parameters

- **input_size** (*int*) – The number of expected features in the input x
- **hidden_size** (*int*) – The number of features in the hidden state h
- **num_layers** (*int*) – Number of recurrent layers.
- **nonlinearity** (*str*) – The non-linearity to use ['tanh'|'relu']. Default: 'tanh'
- **bias** (*bool*) – If False, then the layer does not use bias weights b_ih and b_hh. Default: True
- **batch_first** (*bool*) – If True, then the input and output tensors are provided as (batch, seq, feature)
- **dropout** (*float*) – If non-zero, introduces a dropout layer on the outputs of each RNN layer except the last layer
- **bidirectional** (*bool*) – If True, becomes a bidirectional RNN. Default: False

Inputs: input, mask, h_0

- **input** (seq_len, batch, input_size): tensor containing the features of the input sequence. **mask** (seq_len, batch): 0-1 tensor containing the mask of the input sequence.
- **h_0** (num_layers * num_directions, batch, hidden_size): tensor containing the initial hidden state for each element in the batch.

Outputs: output, h_n

- **output** (seq_len, batch, hidden_size * num_directions): tensor containing the output features (h_k) from the last layer of the RNN, for each k. If a `torch.nn.utils.rnn.PackedSequence` has been given as the input, the output will also be a packed sequence.
- **h_n** (num_layers * num_directions, batch, hidden_size): tensor containing the hidden state for k=seq_len.

class fastNLP.modules.encoder.masked_rnn.**MaskedRNNBase**(*Cell*, *input_size*, *hidden_size*, *num_layers=1*, *bias=True*, *batch_first=False*, *layer_dropout=0*, *step_dropout=0*, *bidirectional=False*, *initial_method=None*, ***kwargs*)

forward(*input*, *mask=None*, *hx=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.

step(*input*, *hx=None*, *mask=None*)
Execute one step forward (only for one-directional RNN).

Parameters

- **input** (*Tensor*) – input tensor of this step. (batch, input_size)
- **hx** (*Tensor*) – the hidden state of last step. (num_layers, batch, hidden_size)
- **mask** (*Tensor*) – the mask tensor of this step. (batch,)

Returns **output** (batch, hidden_size), tensor containing the output of this step from the last layer of RNN. **hn** (num_layers, batch, hidden_size), tensor containing the hidden state of this step

fastNLP.modules.encoder.transformer

class fastNLP.modules.encoder.transformer.**TransformerEncoder**(*num_layers*, ***kargs*)

class SubLayer(*model_size*, *inner_size*, *key_size*, *value_size*, *num_head*, *dropout=0.1*)

forward(*input*, *seq_mask=None*, *atte_mask_out=None*)

Parameters

- **input** – [batch, seq_len, model_size]
- **seq_mask** – [batch, seq_len]

Returns [batch, seq_len, model_size]

forward(*x*, *seq_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.

fastNLP.modules.encoder.variational_rnn

class fastNLP.modules.encoder.variational_rnn.**VarGRU**(**args*, ***kwargs*)
Variational Dropout GRU.

class fastNLP.modules.encoder.variational_rnn.**VarLSTM**(**args*, ***kwargs*)
Variational Dropout LSTM.

class fastNLP.modules.encoder.variational_rnn.**VarRNN**(**args*, ***kwargs*)
Variational Dropout RNN.

class fastNLP.modules.encoder.variational_rnn.**VarRNNBase**(*mode*, *Cell*, *input_size*, *hidden_size*, *num_layers=1*, *bias=True*, *batch_first=False*, *input_dropout=0*, *hidden_dropout=0*, *bidirectional=False*)

Implementation of Variational Dropout RNN network. refer to *A Theoretically Grounded Application of Dropout in Recurrent Neural Networks (Yarin Gal and Zoubin Ghahramani, 2016) https://arxiv.org/abs/1512.05287*.

forward(*input*, *hx=None*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

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

---

class fastNLP.modules.encoder.variational_rnn.**VarRnnCellWrapper**(*cell*, *hidden_size*, *input_p*, *hidden_p*)

Wrapper for normal RNN Cells, make it support variational dropout

forward(*input_x*, *hidden*, *mask_x*, *mask_h*, *is_reversed=False*)

Parameters

- **input_x** (*PackedSequence*) – [seq_len, batch_size, input_size]
- **hidden** – for LSTM, tuple of (h_0, c_0), [batch_size, hidden_size] for other RNN, h_0, [batch_size, hidden_size]
- **mask_x** – [batch_size, input_size] dropout mask for input
- **mask_h** – [batch_size, hidden_size] dropout mask for hidden

Return PackedSequence output [seq_len, bacth_size, hidden_size] hidden: for LSTM, tuple of (h_n, c_n), [batch_size, hidden_size]

for other RNN, h_n, [batch_size, hidden_size]

fastNLP.modules.encoder.variational_rnn.**flip**(*input*, *dims*) → Tensor

Reverse the order of a n-D tensor along given axis in dims.

Args: input (Tensor): the input tensor dims (a list or tuple): axis to flip on

Example:

```
>>> x = torch.arange(8).view(2, 2, 2)
>>> x
tensor([[[ 0,  1],
         [ 2,  3]],

        [[ 4,  5],
         [ 6,  7]]])
>>> torch.flip(x, [0, 1])
tensor([[[ 6,  7],
```

(continues on next page)

(continued from previous page)

```
         [ 4,  5]],

        [[ 2,  3],
         [ 0,  1]]])
```

class fastNLP.modules.encoder.**LSTM**(*input_size*, *hidden_size=100*, *num_layers=1*, *dropout=0.0*, *batch_first=True*, *bidirectional=False*, *bias=True*, *initial_method=None*, *get_hidden=False*)

Long Short Term Memory

Parameters

- **input_size** (*int*) –
- **hidden_size** (*int*) –
- **num_layers** (*int*) –
- **dropout** (*float*) –
- **batch_first** (*bool*) –
- **bidirectional** (*bool*) –
- **bias** (*bool*) –
- **initial_method** (*str*) –
- **get_hidden** (*bool*) –

forward(*x*, *h0=None*, *c0=None*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

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

---

class fastNLP.modules.encoder.**Embedding**(*nums*, *dims*, *padding_idx=0*, *sparse=False*, *init_emb=None*, *dropout=0.0*)

A simple lookup table.

Parameters

- **nums** (*int*) – the size of the lookup table
- **dims** (*int*) – the size of each vector
- **padding_idx** (*int*) – pads the tensor with zeros whenever it encounters this index
- **sparse** (*bool*) – If True, gradient matrix will be a sparse tensor. In this case, only optim.SGD(cuda and cpu) and optim.Adagrad(cpu) can be used

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

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

while the latter silently ignores them.

---

class fastNLP.modules.encoder.**Linear**(*input_size*, *output_size*, *bias=True*, *initial_method=None*)

Parameters

- **input_size** (*int*) – input size
- **output_size** (*int*) – output size
- **bias** (*bool*) –
- **initial_method** (*str*) –

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

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

---

class fastNLP.modules.encoder.**Conv**(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*, *activation='relu'*, *initial_method=None*)

Basic 1-d convolution module, initialized with xavier_uniform.

Parameters

- **in_channels** (*int*) –
- **out_channels** (*int*) –
- **kernel_size** (*tuple*) –
- **stride** (*int*) –
- **padding** (*int*) –
- **dilation** (*int*) –
- **groups** (*int*) –
- **bias** (*bool*) –
- **activation** (*str*) –
- **initial_method** (*str*) –

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

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

---

class fastNLP.modules.encoder.**ConvMaxpool**(*in_channels*, *out_channels*, *kernel_sizes*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*, *activation='relu'*, *initial_method=None*)

Convolution and max-pooling module with multiple kernel sizes.

Parameters

- **in_channels** (*int*) –
- **out_channels** (*int*) –
- **kernel_sizes** (*tuple*) –
- **stride** (*int*) –
- **padding** (*int*) –
- **dilation** (*int*) –
- **groups** (*int*) –
- **bias** (*bool*) –
- **activation** (*str*) –
- **initial_method** (*str*) –

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.

fastNLP.modules.dropout

class fastNLP.modules.dropout.**TimestepDropout**(*p=0.5*, *inplace=False*)

This module accepts a `[batch_size, num_timesteps, embedding_dim)]` and use a single dropout mask of shape `(batch_size, embedding_dim)` to apply on every time step.

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.

fastNLP.modules.other_modules

class fastNLP.modules.other_modules.**BiAffine**(*n_enc*, *n_dec*, *n_labels*, *biaffine=True*, ***kwargs*)

forward(*input_d*, *input_e*, *mask_d=None*, *mask_e=None*)

Parameters

- **input_d** (*Tensor*) – the decoder input tensor with shape = [batch, length_decoder, input_size]
- **input_e** (*Tensor*) – the child input tensor with shape = [batch, length_encoder, input_size]
- **mask_d** – Tensor or None, the mask tensor for decoder with shape = [batch, length_decoder]
- **mask_e** – Tensor or None, the mask tensor for encoder with shape = [batch, length_encoder]

Returns Tensor, the energy tensor with shape = [batch, num_label, length, length]

class fastNLP.modules.other_modules.**BiLinear**(*n_left*, *n_right*, *n_out*, *bias=True*)

forward(*input_left*, *input_right*)

Parameters

- **input_left** (*Tensor*) – the left input tensor with shape = [batch1, batch2, ..., left_features]
- **input_right** (*Tensor*) – the right input tensor with shape = [batch1, batch2, ..., right_features]

class fastNLP.modules.other_modules.**GroupNorm**(*num_features*, *num_groups=20*, *eps=1e-05*)

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

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

---

class fastNLP.modules.other_modules.**LayerNormalization**(*layer_size*, *eps=0.001*)

Parameters

- **layer_size** (*int*) –
- **eps** (*float*) – default=1e-3

forward(*z*)

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.

---

fastNLP.modules.utils

fastNLP.modules.utils.**initial_parameter**(*net*, *initial_method=None*)
A method used to initialize the weights of PyTorch models.

Parameters

- **net** – a PyTorch model
- **initial_method** (*str*) – one of the following initializations.
  - xavier_uniform
  - xavier_normal (default)
  - kaiming_normal, or msra
  - kaiming_uniform
  - orthogonal
  - sparse
  - normal
  - uniform

fastNLP.modules.utils.**seq_mask**(*seq_len*, *max_len*)
Create sequence mask.

Parameters

- **seq_len** – list or torch.Tensor, the lengths of sequences in a batch.
- **max_len** – int, the maximum sequence length in a batch.

Returns mask, torch.LongTensor, [batch_size, max_len]

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

f

Index

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

Z