

---

# langml

*Release 0.0.1*

**Sean Lee**

**Jun 29, 2022**



## GETTING STARTED

<b>1</b>	<b>Installation</b>	<b>3</b>
1.1	From pip . . . . .	3
1.2	From Github . . . . .	3
<b>2</b>	<b>Use langml-cli to quickly train baseline models</b>	<b>5</b>
2.1	Text Classification . . . . .	5
2.2	Named Entity Recognition . . . . .	7
2.3	Contrastive Learning . . . . .	9
2.4	Text Matching . . . . .	10
<b>3</b>	<b>Examples of finetuneing</b>	<b>11</b>
3.1	1. Prepare datasets . . . . .	11
3.2	2. Build models . . . . .	12
3.3	3. Train and Eval . . . . .	12
<b>4</b>	<b>Examples of prompt-based tuning</b>	<b>13</b>
4.1	Prompt-based Classification . . . . .	13
<b>5</b>	<b>How to train PLMs distributedly?</b>	<b>15</b>
<b>6</b>	<b>API Reference</b>	<b>17</b>
6.1	langml . . . . .	17
	<b>Python Module Index</b>	<b>81</b>
	<b>Index</b>	<b>83</b>



LangML (Language ModeL) is a Keras-based and TensorFlow-backend language model toolkit, which provides main-stream pre-trained language models, e.g., BERT/RoBERTa/ALBERT, and their downstream application models.



## INSTALLATION

### 1.1 From pip

You can install or upgrade langml/langml-cli from pip:

```
pip install -U langml
```

### 1.2 From Github

You can also install the latest langml/langml-cli from Github:

```
git clone https://github.com/4AI/langml.git  
cd langml  
python setup.py install
```





## USE LANGML-CLI TO QUICKLY TRAIN BASELINE MODELS

You can use LangML-CLI to train baseline models quickly. You don't need to write any code and just need to prepare the dataset in a specific format.

You can train various baseline models using *langml-cli*:

```
$ langml-cli --help
Usage: langml [OPTIONS] COMMAND [ARGS]...

LangML client

Options:
  --version  Show the version and exit.
  --help     Show this message and exit.

Commands:
  baseline  LangML Baseline client
```

### 2.1 Text Classification

Prepare your data into *JSONLines* format, and provide text and label field in each line, for example:

```
{"text": "this is sentence1", "label": "label1"}
{"text": "this is sentence2", "label": "label2"}
```

#### 1. Bert

```
$ langml-cli baseline clf bert --help
Usage: langml baseline clf bert [OPTIONS]

Options:
  --backbone TEXT          specify backbone: bert | roberta | albert
  --epoch INTEGER          epochs
  --batch_size INTEGER     batch size
  --learning_rate FLOAT    learning rate
  --max_len INTEGER        max len
  --lowercase              do lowercase
  --tokenizer_type TEXT    specify tokenizer type from ['wordpiece',
                           'sentencepiece']
```

(continues on next page)

(continued from previous page)

```

--monitor TEXT          monitor for keras callback
--early_stop INTEGER    patience to early stop
--use_micro              whether to use micro metrics
--config_path TEXT       bert config path [required]
--ckpt_path TEXT         bert checkpoint path [required]
--vocab_path TEXT        bert vocabulary path [required]
--train_path TEXT        train path [required]
--dev_path TEXT          dev path [required]
--test_path TEXT         test path
--save_dir TEXT          dir to save model [required]
--verbose INTEGER        0 = silent, 1 = progress bar, 2 = one line per
                        epoch

--distributed_training   distributed training
--distributed_strategy TEXT distributed training strategy
--help                  Show this message and exit.

```

## 2. BiLSTM

```

$ langml-cli baseline clf bilstm --help
Usage: langml baseline clf bilstm [OPTIONS]

Options:
--epoch INTEGER          epochs
--batch_size INTEGER     batch size
--learning_rate FLOAT    learning rate
--embedding_size INTEGER embedding size
--hidden_size INTEGER    hidden size of lstm
--max_len INTEGER        max len
--lowercase              do lowercase
--tokenizer_type TEXT     specify tokenizer type from ['wordpiece',
                        'sentencepiece']

--monitor TEXT          monitor for keras callback
--early_stop INTEGER    patience to early stop
--use_micro              whether to use micro metrics
--vocab_path TEXT       vocabulary path [required]
--train_path TEXT       train path [required]
--dev_path TEXT         dev path [required]
--test_path TEXT        test path
--save_dir TEXT         dir to save model [required]
--verbose INTEGER        0 = silent, 1 = progress bar, 2 = one line per
                        epoch

--with_attention         apply attention mechanism
--distributed_training   distributed training
--distributed_strategy TEXT distributed training strategy
--help                  Show this message and exit.

```

## 3. TextCNN

```
$ langml-cli baseline clf textcnn --help
```

(continues on next page)

(continued from previous page)

Usage: langml baseline clf textcnn [OPTIONS]

Options:

```

--epoch INTEGER           epochs
--batch_size INTEGER      batch size
--learning_rate FLOAT     learning rate
--embedding_size INTEGER  embedding size
--filter_size INTEGER     filter size of convolution
--max_len INTEGER         max len
--lowercase               do lowercase
--tokenizer_type TEXT     specify tokenizer type from ['wordpiece',
                          'sentencepiece']

--monitor TEXT           monitor for keras callback
--early_stop INTEGER     patience to early stop
--use_micro              whether to use micro metrics
--vocab_path TEXT        vocabulary path [required]
--train_path TEXT        train path [required]
--dev_path TEXT          dev path [required]
--test_path TEXT         test path
--save_dir TEXT          dir to save model [required]
--verbose INTEGER        0 = silent, 1 = progress bar, 2 = one line per
                          epoch

--distributed_training    distributed training
--distributed_strategy TEXT distributed training strategy
--help                   Show this message and exit.

```

## 2.2 Named Entity Recognition

Prepare your data in the following format:

use “t” to separate entity segment and entity type in a sentence, and use “nn” to separate different sentences.

An English example:

```

I like    0
apples   Fruit

I like    0
pineapples Fruit

```

A Chinese example:

```

0
  LOC

0
  LOC

```

1) BERT-CRF

```
$ langml-cli baseline ner bert-crf --help
Usage: langml baseline ner bert-crf [OPTIONS]

Options:
  --backbone TEXT          specify backbone: bert | roberta | albert
  --epoch INTEGER          epochs
  --batch_size INTEGER     batch size
  --learning_rate FLOAT    learning rate
  --dropout_rate FLOAT     dropout rate
  --max_len INTEGER        max len
  --lowercase              do lowercase
  --tokenizer_type TEXT    specify tokenizer type from ['wordpiece',
                           'sentencepiece']
  --config_path TEXT       bert config path [required]
  --ckpt_path TEXT         bert checkpoint path [required]
  --vocab_path TEXT        bert vocabulary path [required]
  --train_path TEXT        train path [required]
  --dev_path TEXT          dev path [required]
  --test_path TEXT         test path
  --save_dir TEXT          dir to save model [required]
  --monitor TEXT           monitor for keras callback
  --early_stop INTEGER     patience to early stop
  --verbose INTEGER        0 = silent, 1 = progress bar, 2 = one line per
                           epoch
  --distributed_training    distributed training
  --distributed_strategy TEXT distributed training strategy
  --help                   Show this message and exit.
```

## 2) LSTM-CRF

```
$ langml-cli baseline ner lstm-crf --help
Usage: langml baseline ner lstm-crf [OPTIONS]

Options:
  --epoch INTEGER          epochs
  --batch_size INTEGER     batch size
  --learning_rate FLOAT    learning rate
  --dropout_rate FLOAT     dropout rate
  --embedding_size INTEGER embedding size
  --hidden_size INTEGER    hidden size
  --max_len INTEGER        max len
  --lowercase              do lowercase
  --tokenizer_type TEXT    specify tokenizer type from ['wordpiece',
                           'sentencepiece']
  --vocab_path TEXT        vocabulary path [required]
  --train_path TEXT        train path [required]
  --dev_path TEXT          dev path [required]
  --test_path TEXT         test path
  --save_dir TEXT          dir to save model [required]
  --monitor TEXT           monitor for keras callback
  --early_stop INTEGER     patience to early stop
  --verbose INTEGER        0 = silent, 1 = progress bar, 2 = one line per
                           epoch
```

(continues on next page)

(continued from previous page)

```
--distributed_training    distributed training
--distributed_strategy TEXT distributed training strategy
--help                    Show this message and exit.
```

## 2.3 Contrastive Learning

Prepare your data into *JSONLines* format:

a) for evaluation, should include *text\_left*, *text\_right*, and *label* fields

```
{"text_left": "text left1", "text_right": "text right1", "label": "0/1"}
{"text_left": "text left1", "text_right": "text right2", "label": "0/1"}
```

b) no need to evaluate, just provide *text* field.

```
{"text": "this is a text1"}
{"text": "this is a text2"}
```

1. simcse

```
$ langml-cli baseline contrastive simcse --help
Usage: langml baseline contrastive simcse [OPTIONS]

Options:
  --backbone TEXT          specify backbone: bert | roberta | albert
  --epoch INTEGER          epochs
  --batch_size INTEGER     batch size
  --learning_rate FLOAT    learning rate
  --dropout_rate FLOAT     dropout rate
  --temperature FLOAT      temperature
  --pooling_strategy TEXT   specify pooling_strategy from ["cls", "first-
                             last-avg", "last-avg"]
  --max_len INTEGER        max len
  --early_stop INTEGER     patience of early stop
  --monitor TEXT           metrics monitor
  --lowercase              do lowercase
  --tokenizer_type TEXT     specify tokenizer type from ['wordpiece',
                             'sentencepiece']
  --config_path TEXT       bert config path [required]
  --ckpt_path TEXT         bert checkpoint path [required]
  --vocab_path TEXT        bert vocabulary path [required]
  --train_path TEXT        train path [required]
  --test_path TEXT         test path
  --save_dir TEXT          dir to save model [required]
  --verbose INTEGER        0 = silent, 1 = progress bar, 2 = one line per
                             epoch
  --apply_aeda             apply AEDA to augment data
  --aeda_language TEXT     specify AEDA language, ["EN", "CN"]
  --do_evaluate            do evaluation
  --distributed_training    distributed training
```

(continues on next page)

(continued from previous page)

```
--distributed_strategy TEXT distributed training strategy
--help                      Show this message and exit.
```

## 2.4 Text Matching

Prepare your data into *JSONLines* format, three fields *text\_left*, *text\_right*, and *label* are required.

```
{"text_left": "text left1", "text_right": "text right1", "label": "label1"}
{"text_left": "text left1", "text_right": "text right2", "label": "label2"}
```

1. sentence bert

For the regression task, the label should be a float value or an integer. For the classification task, the label should be an integer or a string value.

```
$ langml-cli baseline matching sbert --help

Usage: langml baseline matching sbert [OPTIONS]

Options:
  --backbone TEXT          specify backbone: bert | roberta | albert
  --epoch INTEGER          epochs
  --batch_size INTEGER     batch size
  --learning_rate FLOAT    learning rate
  --dropout_rate FLOAT     dropout rate
  --task TEXT              specify task from ["regression",
                           "classification"]
  --pooling_strategy TEXT  specify pooling_strategy from ["cls", "mean",
                           "max"]
  --max_len INTEGER        max len
  --early_stop INTEGER     patience of early stop
  --monitor TEXT           metrics monitor
  --lowercase              do lowercase
  --tokenizer_type TEXT    specify tokenizer type from ['wordpiece',
                           'sentencepiece']
  --config_path TEXT       bert config path [required]
  --ckpt_path TEXT         bert checkpoint path [required]
  --vocab_path TEXT        bert vocabulary path [required]
  --train_path TEXT        train path [required]
  --dev_path TEXT          dev path [required]
  --test_path TEXT         test path
  --save_dir TEXT          dir to save model [required]
  --verbose INTEGER        0 = silent, 1 = progress bar, 2 = one line per
                           epoch

  --distributed_training    distributed training
  --distributed_strategy TEXT distributed training strategy
  --help                   Show this message and exit.
```

## EXAMPLES OF FINETUNEING

To finetune a model, you need to prepare pretrained language models (PLMs). Currently, LangML supports BERT/RoBERTa/ALBERT PLMs. You can download PLMs from [google-research/bert](#), [google-research/albert](#), [Chinese RoBERTa](#) etc.

### 3.1 1. Prepare datasets

You need to use specific tokenizers in terms of PLMs to initialize a tokenizer and convert texts to vocabulary indices. LangML wraps [huggingface/tokenizers](#) and [google/sentencepiece](#) to provide a uniform interface. Specifically, you can initialize a WordPiece tokenizer via `langml.tokenizer.WPTokenizer`, and initialize a sentencepiece tokenizer via `langml.tokenizer.SPTokenizer`.

```
from langml import keras, L
from langml.tokenizer import WPTokenizer

vocab_path = '/path/to/vocab.txt'
tokenizer = WPTokenizer(vocab_path)
# specify max token length
tokenizer.enable_truncation(max_length=512)

class DataLoader:
    def __init__(self, tokenizer):
        # define initializer here
        self.tokenizer = tokenizer

    def __iter__(self, data):
        # define your data generator here
        for text, label in data:
            tokenized = self.tokenizer.encode(text)
            token_ids = tokenized.ids
            segment_ids = tokenized.segment_ids
            # ...
```

## 3.2 2. Build models

You can use `langml.plm.load_bert` to load a BERT/RoBERTa model, and use `langml.plm.load_albert` to load an ALBERT model.

```
from langml import keras, L
from langml.plm import load_bert

config_path = '/path/to/bert_config.json'
ckpt_path = '/path/to/bert_model.ckpt'
vocab_path = '/path/to/vocab.txt'

bert_model, bert_instance = load_bert(config_path, ckpt_path)
# get CLS representation
cls_output = L.Lambda(lambda x: x[:, 0])(bert_model.output)
output = L.Dense(2, activation='softmax',
                 kernel_initializer=bert_instance.initializer)(cls_output)
train_model = keras.Model(bert_model.input, cls_output)
train_model.summary()
train_model.compile(loss='categorical_crossentropy', optimizer=keras.optimizer.Adam(1e-5))
```

## 3.3 3. Train and Eval

After defining the data loader and model, you can train and evaluate your model as most Keras models do.



## EXAMPLES OF PROMPT-BASED TUNING

Prompt-based tuning is the latest paradigm to adapt PLMs to downstream NLP tasks, which embeds a textual template into the input text and directly uses the MLM task of PLMs to train models.

Currently support:

- PTuning: GPT Understands, Too

### 4.1 Prompt-based Classification

There are three steps to build a prompt-based classifier.

1. Define a template

```
from langml.prompt import Template
from langml.tokenizer import WPTokenizer

vocab_path = '/path/to/vocab.txt'

tokenizer = WPTokenizer(vocab_path, lowercase=True)
template = Template(
    # must specify tokens that are defined in the vocabulary, and the mask token is.
    ↪required
    template=['it', 'was', '[MASK]', '.'],
    # must specify tokens that are defined in the vocabulary.
    label_tokens_map={
        'positive': ['good'],
        'negative': ['bad', 'terrible']
    },
    tokenizer=tokenizer
)
```

2. Define a prompt-based model

```
from langml.prompt import PTuningPrompt, PTuningForClassification

bert_config_path = '/path/to/bert_config.json'
bert_ckpt_path = '/path/to/bert_model.ckpt'

prompt_model = PTuningPrompt('bert', bert_config_path, bert_ckpt_path,
```

(continues on next page)

(continued from previous page)

```
template, freeze_plm=False, learning_rate=5e-5, encoder=
↪ 'lstm')
prompt_classifier = PTuningForClassification(prompt_model, tokenizer)
```

### 3. Train on dataset

```
data = [('I do not like this food', 'negative'),
        ('I hate you', 'negative'),
        ('I like you', 'positive'),
        ('I like this food', 'positive')]

X = [d for d, _ in data]
y = [l for _, l in data]

prompt_classifier.fit(X, y, X, y, batch_size=2, epoch=50, model_path='best_model.weight')
# load pretrained model
# prompt_classifier.load('best_model.weight')
print("pred", prompt_classifier.predict('I hate you'))
```

For more examples visit [langml/examples](https://langml.com/examples)

## HOW TO TRAIN PLMS DISTRIBUTEDLY?

To train distributedly, you need to use *tensorflow.keras*. First, you need to define an environment variable *TF\_KERAS* and assign *1* to it, for example, *export TF\_KERAS=1* for Linux. Then manually restore PLMs weights after model compiling, as follows:

```
from langml import keras, L
from langml.plm import load_bert

config_path = '/path/to/bert_config.json'
ckpt_path = '/path/to/bert_model.ckpt'
vocab_path = '/path/to/vocab.txt'

# lazy restore
bert_model, bert_instance, restore_weight_callback = load_bert(config_path, ckpt_path,
↳ lazy_restore=True)
# get CLS representation
cls_output = L.Lambda(lambda x: x[:, 0])(bert_model.output)
output = L.Dense(2, activation='softmax',
                  kernel_initializer=bert_instance.initializer)(cls_output)
train_model = keras.Model(bert_model.input, cls_output)
train_model.summary()
train_model.compile(loss='categorical_crossentropy', optimizer=keras.optimizer.Adam(1e-
↳ 5))
# restore weights
restore_weight_callback(bert_model)
```



## API REFERENCE

This page contains auto-generated API reference documentation<sup>1</sup>.

### 6.1 langml

#### 6.1.1 Subpackages

`langml.baselines`

##### Subpackages

`langml.baselines.clf`

##### Submodules

`langml.baselines.clf.bert`

##### Module Contents

##### Classes

---

*BertClassifier*

---

```
class langml.baselines.clf.bert.BertClassifier(config_path: str, ckpt_path: str, params:
                                             langml.baselines.Parameters, backbone: str =
                                             'roberta')
```

Bases: `langml.baselines.BaselineModel`

**build\_model**(self, lazy\_restore=False) → langml.tensor\_typing.Models

---

<sup>1</sup> Created with sphinx-autoapi

langml.baselines.clf.bilstm

## Module Contents

### Classes

---

*BiLSTMClassifier*

---

**class** langml.baselines.clf.bilstm.**BiLSTMClassifier**(*params*: langml.baselines.Parameters,  
with\_attention: bool = False)

Bases: langml.baselines.BaselineModel

**build\_model**(*self*) → langml.tensor\_typing.Models

langml.baselines.clf.cli

## Module Contents

## Functions

---

```
train(model_instance: object, params:
langml.baselines.Parameters, epoch: int, save_dir:
str, train_path: str, dev_path: str, test_path: str, vo-
cab_path: str, tokenizer_type: str, lowercase: bool,
max_len: int, batch_size: int, distributed_training:
bool, distributed_strategy: str, use_micro: bool,
monitor: str, early_stop: int, verbose: int)
```

---

```
clf()
```

```
classification command line tools
```

---

```
bert(backbone: str, epoch: int, batch_size: int,
learning_rate: float, max_len: Optional[int], lower-
case: bool, tokenizer_type: Optional[str], monitor:
str, early_stop: int, use_micro: bool, config_path:
str, ckpt_path: str, vocab_path: str, train_path: str,
dev_path: str, test_path: str, save_dir: str, verbose: int,
distributed_training: bool, distributed_strategy: str)
```

---

```
textcnn(epoch: int, batch_size: int, learning_rate:
float, embedding_size: int, filter_size: int, max_len:
Optional[int], lowercase: bool, tokenizer_type: Op-
tional[str], monitor: str, early_stop: int, use_micro:
bool, vocab_path: str, train_path: str, dev_path:
str, test_path: str, save_dir: str, verbose: int, dis-
tributed_training: bool, distributed_strategy: str)
```

---

```
bilstm(epoch: int, batch_size: int, learning_rate:
float, embedding_size: int, hidden_size: int, max_len:
Optional[int], lowercase: bool, tokenizer_type: Op-
tional[str], monitor: str, early_stop: int, use_micro:
bool, vocab_path: str, train_path: str, dev_path: str,
test_path: str, save_dir: str, verbose: int, with_attention:
bool, distributed_training: bool, distributed_strategy:
str)
```

---

```
langml.baselines.clf.cli.train(model_instance: object, params: langml.baselines.Parameters, epoch: int,
save_dir: str, train_path: str, dev_path: str, test_path: str, vocab_path: str,
tokenizer_type: str, lowercase: bool, max_len: int, batch_size: int,
distributed_training: bool, distributed_strategy: str, use_micro: bool,
monitor: str, early_stop: int, verbose: int)
```

```
langml.baselines.clf.cli.clf()
classification command line tools
```

```
langml.baselines.clf.cli.bert(backbone: str, epoch: int, batch_size: int, learning_rate: float, max_len:
Optional[int], lowercase: bool, tokenizer_type: Optional[str], monitor: str,
early_stop: int, use_micro: bool, config_path: str, ckpt_path: str,
vocab_path: str, train_path: str, dev_path: str, test_path: str, save_dir: str,
verbose: int, distributed_training: bool, distributed_strategy: str)
```

```
langml.baselines.clf.cli.textcnn(epoch: int, batch_size: int, learning_rate: float, embedding_size: int,
filter_size: int, max_len: Optional[int], lowercase: bool, tokenizer_type:
Optional[str], monitor: str, early_stop: int, use_micro: bool, vocab_path:
str, train_path: str, dev_path: str, test_path: str, save_dir: str, verbose:
int, distributed_training: bool, distributed_strategy: str)
```

```
langml.baselines.clf.cli.bilstm(epoch: int, batch_size: int, learning_rate: float, embedding_size: int,
                                hidden_size: int, max_len: Optional[int], lowercase: bool, tokenizer_type:
                                Optional[str], monitor: str, early_stop: int, use_micro: bool, vocab_path:
                                str, train_path: str, dev_path: str, test_path: str, save_dir: str, verbose: int,
                                with_attention: bool, distributed_training: bool, distributed_strategy: str)
```

**langml.baselines.clf.dataloader**

## Module Contents

### Classes

---

*DataLoader*

---

*TFDataLoader*

---

```
class langml.baselines.clf.dataloader.DataLoader(data: List, tokenizer: object, label2id: Dict,
                                                batch_size: int = 32, is_bert: bool = True)
```

Bases: *langml.baselines.BaseDataLoader*

**\_\_len\_\_**(self) → int

**static load\_data**(fpath: str, build\_vocab: bool = False) → List

**make\_iter**(self, random: bool = False)

```
class langml.baselines.clf.dataloader.TFDataLoader(data: List, tokenizer: object, label2id: Dict,
                                                  batch_size: int = 32, is_bert: bool = True)
```

Bases: *DataLoader*

**make\_iter**(self, random: bool = False)

**\_\_call\_\_**(self, random: bool = False)

**langml.baselines.clf.textcnn**

## Module Contents

### Classes

---

*TextCNNClassifier*

---

```
class langml.baselines.clf.textcnn.TextCNNClassifier(params: langml.baselines.Parameters)
```

Bases: *langml.baselines.BaselineModel*

**build\_model**(self) → langml.tensor\_typing.Models



## Package Contents

### Classes

---

*Infer*

---

### Functions

---

*compute\_detail\_metrics*(infer: object, datas: List,  
use\_micro=False) → Tuple[float, float, Union[str, Dict]]

---

### Attributes

---

*TF\_VERSION*

---



---

*Models*

---

langml.baselines.clf.**TF\_VERSION**

langml.baselines.clf.**Models**

**class** langml.baselines.clf.**Infer**(model: langml.tensor\_typing.Models, tokenizer: object, id2label: Dict,  
is\_bert: bool = True)

    \_\_call\_\_(self, text: str)

langml.baselines.clf.**compute\_detail\_metrics**(infer: object, datas: List, use\_micro=False) → Tuple[float,  
float, Union[str, Dict]]

langml.baselines.contrastive

### Subpackages

langml.baselines.contrastive.simcse

### Submodules

langml.baselines.contrastive.simcse.dataloder

### Module Contents

## Classes

---

*DataLoader*

---

---

*TFDataLoader*

---

```
class langml.baselines.contrastive.simcse.dataloder.DataLoader(data: List, tokenizer: object,  
                                                             batch_size: int = 32)
```

Bases: *langml.baselines.BaseDataLoader*

`__len__(self)` → int

**static load\_data**(*fpath: str, apply\_aeda: bool = True, aeda\_tokenize: Callable = whitespace\_tokenize,*  
 *aeda\_language: str = 'EN')* → Tuple[List[Tuple[str, str]], List[Tuple[str, str, int]]]

### Parameters

- **fpath** – str, path of data
- **apply\_aeda** – bool, whether to apply the AEDA technique to augment data, default True
- **aeda\_tokenize** – Callable, specify aeda tokenize function, it works when set apply\_aeda=True
- **aeda\_language** – str, specifying the language, it works when set apply\_aeda=True

**make\_iter**(*self, random: bool = False*)

```
class langml.baselines.contrastive.simcse.dataloder.TFDataLoader(data: List, tokenizer: object,  
                                                                batch_size: int = 32)
```

Bases: *DataLoader*

**make\_iter**(*self, random: bool = False*)

**\_\_call\_\_**(*self, random: bool = False*)

**langml.baselines.contrastive.simcse.model**

## Module Contents

### Classes

---

*SimCSE*

---

## Functions

---

*simcse\_loss*(y\_true, y\_pred)

---

`langml.baselines.contrastive.simcse.model.simcse_loss(y_true, y_pred)`

**class** `langml.baselines.contrastive.simcse.model.SimCSE`(*config\_path: str, ckpt\_path: str, params: langml.baselines.Parameters, backbone: str = 'roberta'*)

Bases: *langml.baselines.BaselineModel*

**get\_pooling\_output**(*self, model: langml.tensor\_typing.Models, output\_index: int, pooling\_strategy: str = 'cls'*) → `langml.tensor_typing.Tensors`

get pooling output :param model: keras.Model, BERT model :param output\_index: int, specify output index of feedforward layer. :param pooling\_strategy: str, specify pooling strategy from ['cls', 'first-last-avg', 'last-avg'], default *cls*

**build\_model**(*self, pooling\_strategy: str = 'cls', lazy\_restore: bool = False*) → `langml.tensor_typing.Models`

## Package Contents

### Classes

---

*DataLoader*

---



---

*TFDataLoader*

---



---

*SimCSE*

---

**class** `langml.baselines.contrastive.simcse.DataLoader`(*data: List, tokenizer: object, batch\_size: int = 32*)

Bases: *langml.baselines.BaseDataLoader*

**\_\_len\_\_**(*self*) → `int`

**static load\_data**(*fpath: str, apply\_aeda: bool = True, aeda\_tokenize: Callable = whitespace\_tokenize, aeda\_language: str = 'EN'*) → `Tuple[List[Tuple[str, str]], List[Tuple[str, str, int]]]`

#### Parameters

- **fpath** – str, path of data
- **apply\_aeda** – bool, whether to apply the AEDA technique to augment data, default True
- **aeda\_tokenize** – Callable, specify aeda tokenize function, it works when set `apply_aeda=True`
- **aeda\_language** – str, specifying the language, it works when set `apply_aeda=True`

**make\_iter**(*self, random: bool = False*)

```
class langml.baselines.contrastive.simcse.TFDataLoader(data: List, tokenizer: object, batch_size: int  
                                                    = 32)
```

Bases: [DataLoader](#)

```
make_iter(self, random: bool = False)
```

```
__call__(self, random: bool = False)
```

```
class langml.baselines.contrastive.simcse.SimCSE(config_path: str, ckpt_path: str, params:  
                                              langml.baselines.Parameters, backbone: str =  
                                              'roberta'))
```

Bases: [langml.baselines.BaselineModel](#)

```
get_pooling_output(self, model: langml.tensor_typing.Models, output_index: int, pooling_strategy: str =  
                  'cls')  $\rightarrow$  langml.tensor_typing.Tensors
```

get pooling output :param model: keras.Model, BERT model :param output\_index: int, specify output index of feedforward layer. :param pooling\_strategy: str, specify pooling strategy from ['cls', 'first-last-avg', 'last-avg'], default *cls*

```
build_model(self, pooling_strategy: str = 'cls', lazy_restore: bool = False)  $\rightarrow$  langml.tensor_typing.Models
```

## Submodules

[langml.baselines.contrastive.cli](#)

## Module Contents

### Functions

<a href="#">contrastive()</a>	contrastive learning command line tools
<hr/>	
<pre><a href="#">simcse</a>(<i>backbone: str, epoch: int, batch_size: int,</i> <i>learning_rate: float, dropout_rate: float, tempera-</i> <i>ture: float, pooling_strategy: str, max_len: Op-</i> <i>tional[int], early_stop: int, monitor: str, lowercase:</i> <i>bool, tokenizer_type: Optional[str], config_path: str,</i> <i>ckpt_path: str, vocab_path: str, train_path: str,</i> <i>test_path: str, save_dir: str, verbose: int, apply_aeda:</i> <i>bool, aeda_language: str, do_evaluate: bool, dis-</i> <i>tributed_training: bool, distributed_strategy: str</i>)</pre> <hr/>	
<pre>langml.baselines.contrastive.cli.<a href="#">contrastive</a>()</pre> <p>contrastive learning command line tools</p>	
<pre>langml.baselines.contrastive.cli.<a href="#">simcse</a>(<i>backbone: str, epoch: int, batch_size: int, learning_rate: float,</i> <i>dropout_rate: float, temperature: float, pooling_strategy: str,</i> <i>max_len: Optional[int], early_stop: int, monitor: str, lowercase:</i> <i>bool, tokenizer_type: Optional[str], config_path: str, ckpt_path:</i> <i>str, vocab_path: str, train_path: str, test_path: str, save_dir: str,</i> <i>verbose: int, apply_aeda: bool, aeda_language: str,</i> <i>do_evaluate: bool, distributed_training: bool,</i> <i>distributed_strategy: str</i>)</pre>	

`langml.baselines.contrastive.utils`

## Module Contents

### Functions

---

<code>aeda_augment</code> (words: List[str], ratio: float = 0.3, language: str = 'EN') → str	AEDAAAn Easier Data Augmentation Technique for Text Classification
<code>whitespace_tokenize</code> (text: str) → List[str]	

---

### Attributes

---

`CN_PUNCTUATIONS`

---



---

`EN_PUNCTUATIONS`

---

`langml.baselines.contrastive.utils.CN_PUNCTUATIONS` = [' ', '!', '!', '!', '!', '!']

`langml.baselines.contrastive.utils.EN_PUNCTUATIONS` = ['.', ',', '!', '?', ';', ':']

`langml.baselines.contrastive.utils.aeda_augment`(words: List[str], ratio: float = 0.3, language: str = 'EN') → str

AEDAAAn Easier Data Augmentation Technique for Text Classification :param text: str, input text :param ratio: float, ratio to add punctuation randomly :param language: str, specify language from ['EN', 'CN'], default EN

`langml.baselines.contrastive.utils.whitespace_tokenize`(text: str) → List[str]

`langml.baselines.matching`

## Subpackages

`langml.baselines.matching.sbert`

## Submodules

`langml.baselines.matching.sbert.dataloder`

## Module Contents

### Classes

---

`DataLoader`

---



---

`TFDataLoader`

---

```
class langml.baselines.matching.sbert.dataloder.DataLoader(data: List, tokenizer: object,
                                                           batch_size: int = 32)
```

Bases: [langml.baselines.BaseDataLoader](#)

```
__len__(self) → int
```

```
static load_data(fpath: str, build_vocab: bool = False, label2idx: Optional[Dict] = None) →
Union[List[Tuple[str, str, int]], Tuple[List[Tuple[str, str, int]], Dict]]
```

#### Parameters

- **fpath** – str, path of data
- **build\_vocab** – bool, whether to build vocabulary
- **label2idx** – Optional[Dict], label to index dict

```
make_iter(self, random: bool = False)
```

```
class langml.baselines.matching.sbert.dataloder.TFDataLoader(data: List, tokenizer: object,
                                                             batch_size: int = 32)
```

Bases: [DataLoader](#)

```
make_iter(self, random: bool = False)
```

```
__call__(self, random: bool = False)
```

[langml.baselines.matching.sbert.model](#)

## Module Contents

### Classes

---

[SentenceBert](#)

---

```
class langml.baselines.matching.sbert.model.SentenceBert(config_path: str, ckpt_path: str, params:
                                                         langml.baselines.Parameters, backbone:
                                                         str = 'roberta')
```

Bases: [langml.baselines.BaselineModel](#)

```
get_pooling_output(self, model: langml.tensor_typing.Models, output_index: int, pooling_strategy: str =
'cls') → langml.tensor_typing.Tensors
```

get pooling output :param model: keras.Model, BERT model :param output\_index: int, specify output index of feedforward layer. :param pooling\_strategy: str, specify pooling strategy from ['cls', 'first-last-avg', 'last-avg'], default *cls*

```
build_model(self, task: str = 'regression', pooling_strategy: str = 'cls', lazy_restore: bool = False) →
langml.tensor_typing.Models
```

## Package Contents

### Classes

---

*DataLoader*

---

*TFDataLoader*

---

*SentenceBert*

---

**class** langml.baselines.matching.sbert.**DataLoader**(data: List, tokenizer: object, batch\_size: int = 32)

Bases: [langml.baselines.BaseDataLoader](#)

**\_\_len\_\_**(self) → int

**static load\_data**(fpath: str, build\_vocab: bool = False, label2idx: Optional[Dict] = None) → Union[List[Tuple[str, str, int]], Tuple[List[Tuple[str, str, int]], Dict]]

#### Parameters

- **fpath** – str, path of data
- **build\_vocab** – bool, whether to build vocabulary
- **label2idx** – Optional[Dict], label to index dict

**make\_iter**(self, random: bool = False)

**class** langml.baselines.matching.sbert.**TFDataLoader**(data: List, tokenizer: object, batch\_size: int = 32)

Bases: [DataLoader](#)

**make\_iter**(self, random: bool = False)

**\_\_call\_\_**(self, random: bool = False)

**class** langml.baselines.matching.sbert.**SentenceBert**(config\_path: str, ckpt\_path: str, params: [langml.baselines.Parameters](#), backbone: str = 'roberta')

Bases: [langml.baselines.BaselineModel](#)

**get\_pooling\_output**(self, model: langml.tensor\_typing.Models, output\_index: int, pooling\_strategy: str = 'cls') → langml.tensor\_typing.Tensors

get pooling output :param model: keras.Model, BERT model :param output\_index: int, specify output index of feedforward layer. :param pooling\_strategy: str, specify pooling strategy from ['cls', 'first-last-avg', 'last-avg'], default *cls*

**build\_model**(self, task: str = 'regression', pooling\_strategy: str = 'cls', lazy\_restore: bool = False) → langml.tensor\_typing.Models

## Submodules

`langml.baselines.matching.cli`

## Module Contents

### Functions

---

<code>matching()</code>	text matching command line tools
-------------------------	----------------------------------

---

<code>sbert</code> (backbone: str, epoch: int, batch_size: int, learning_rate: float, dropout_rate: float, task: str, pooling_strategy: str, max_len: Optional[int], early_stop: int, monitor: str, lowercase: bool, tokenizer_type: Optional[str], config_path: str, ckpt_path: str, vocab_path: str, train_path: str, dev_path: str, test_path: str, save_dir: str, verbose: int, distributed_training: bool, distributed_strategy: str)	
--	--

---

`langml.baselines.matching.cli.matching()`

text matching command line tools

`langml.baselines.matching.cli.sbert`(*backbone: str, epoch: int, batch\_size: int, learning\_rate: float, dropout\_rate: float, task: str, pooling\_strategy: str, max\_len: Optional[int], early\_stop: int, monitor: str, lowercase: bool, tokenizer\_type: Optional[str], config\_path: str, ckpt\_path: str, vocab\_path: str, train\_path: str, dev\_path: str, test\_path: str, save\_dir: str, verbose: int, distributed\_training: bool, distributed\_strategy: str*)

`langml.baselines.ner`

## Submodules

`langml.baselines.ner.bert_crf`

## Module Contents

### Classes

---

<code>BertCRF</code>	
----------------------	--

---

<code>class langml.baselines.ner.bert_crf.BertCRF</code> ( <i>config_path: str, ckpt_path: str, params: langml.baselines.Parameters, backbone: str = 'roberta'</i> )	
Bases: <code>langml.baselines.BaselineModel</code>	
<code>build_model</code> ( <i>self, lazy_restore=False</i> ) → <code>langml.tensor_typing.Models</code>	

---



`langml.baselines.ner.cli`

## Module Contents

## Functions

---

```
train(model_instance: object, params:
langml.baselines.Parameters, epoch: int, save_dir:
str, train_path: str, dev_path: str, test_path: str, vo-
cab_path: str, tokenizer_type: str, lowercase: bool,
max_len: int, batch_size: int, distributed_training:
bool, distributed_strategy: str, monitor: str, early_stop:
int, verbose: int)
```

---

```
ner() ner command line tools
```

---

```
bert_crf(backbone: str, epoch: int, batch_size: int,
learning_rate: float, dropout_rate: float, max_len:
Optional[int], lowercase: bool, tokenizer_type: Op-
tional[str], config_path: str, ckpt_path: str, vocab_path:
str, train_path: str, dev_path: str, test_path: str, save_dir:
str, monitor: str, early_stop: int, verbose: int, dis-
tributed_training: bool, distributed_strategy: str)
```

---

```
lstm_crf(epoch: int, batch_size: int, learning_rate:
float, dropout_rate: float, embedding_size: int, hid-
den_size: int, max_len: Optional[int], lowercase:
bool, tokenizer_type: Optional[str], vocab_path: str,
train_path: str, dev_path: str, test_path: str, save_dir:
str, monitor: str, early_stop: int, verbose: int, dis-
tributed_training: bool, distributed_strategy: str)
```

---

```
langml.baselines.ner.cli.train(model_instance: object, params: langml.baselines.Parameters, epoch: int,
save_dir: str, train_path: str, dev_path: str, test_path: str, vocab_path: str,
tokenizer_type: str, lowercase: bool, max_len: int, batch_size: int,
distributed_training: bool, distributed_strategy: str, monitor: str,
early_stop: int, verbose: int)
```

```
langml.baselines.ner.cli.ner()
    ner command line tools
```

```
langml.baselines.ner.cli.bert_crf(backbone: str, epoch: int, batch_size: int, learning_rate: float,
dropout_rate: float, max_len: Optional[int], lowercase: bool,
tokenizer_type: Optional[str], config_path: str, ckpt_path: str,
vocab_path: str, train_path: str, dev_path: str, test_path: str, save_dir:
str, monitor: str, early_stop: int, verbose: int, distributed_training:
bool, distributed_strategy: str)
```

```
langml.baselines.ner.cli.lstm_crf(epoch: int, batch_size: int, learning_rate: float, dropout_rate: float,
embedding_size: int, hidden_size: int, max_len: Optional[int],
lowercase: bool, tokenizer_type: Optional[str], vocab_path: str,
train_path: str, dev_path: str, test_path: str, save_dir: str, monitor: str,
early_stop: int, verbose: int, distributed_training: bool,
distributed_strategy: str)
```

## langml.baselines.ner.dataloader

### Module Contents

#### Classes

---

*DataLoader*

---

---

*TFDataLoader*

---

```
class langml.baselines.ner.dataloader.DataLoader(data: List, tokenizer: object, label2id: Dict,  
                                              batch_size: int = 32, max_len: Optional[int] =  
                                              None, is_bert: bool = True)
```

Bases: *langml.baselines.BaseDataLoader*

```
encode_data(self, data: List[Tuple[str, str]]) → Tuple[List[int], List[int], List[int]]
```

```
static load_data(fpath: str, build_vocab: bool = False) → List
```

```
__len__(self) → int
```

```
make_iter(self, random: bool = False)
```

```
class langml.baselines.ner.dataloader.TFDataLoader(data: List, tokenizer: object, label2id: Dict,  
                                                  batch_size: int = 32, max_len: Optional[int] =  
                                                  None, is_bert: bool = True)
```

Bases: *DataLoader*

```
make_iter(self, random: bool = False)
```

```
__call__(self, random: bool = False)
```

## langml.baselines.ner.lstm\_crf

### Module Contents

#### Classes

---

*LSTMCRF*

---

```
class langml.baselines.ner.lstm_crf.LSTMCRF(params: langml.baselines.Parameters)
```

Bases: *langml.baselines.BaselineModel*

```
build_model(self) → langml.tensor_typing.Models
```

## Package Contents

### Classes

---

*Infer*

---

### Functions

---

*bio\_decode*(tags: List[str]) → List[Tuple[int, int, str]]    Decode BIO tags

---

*compute\_detail\_metrics*(model:  
langml.tensor\_typing.Models,    dataloader:    object,  
id2label: Dict, is\_bert: bool = True)

---

### Attributes

---

*TF\_VERSION*

---

*Models*

---

*re\_split*

---

langml.baselines.ner.**TF\_VERSION**

langml.baselines.ner.**bio\_decode**(tags: List[str]) → List[Tuple[int, int, str]]

Decode BIO tags

Examples: >>> bio\_decode(['B-PER', 'I-PER', 'O', 'B-ORG', 'I-ORG', 'I-ORG']) >>> [(0, 1, 'PER'), (3, 5, 'ORG')]

langml.baselines.ner.**Models**

langml.baselines.ner.**re\_split**

**class** langml.baselines.ner.**Infer**(model: langml.tensor\_typing.Models, tokenizer: object, id2label: Dict, max\_chunk\_len: Optional[int] = None, is\_bert: bool = True)

**decode\_one**(self, text: str, base\_position: int = 0)

#### Parameters

- **text** (-) – str
- **base\_position** (-) – int

#### Returns

[(entity, start, end, entity\_type)]

#### Return type

list of tuple

```
__call__(self, text: str)
```

```
langml.baselines.ner.compute_detail_metrics(model: langml.tensor_typing.Models, data_loader: object,
                                             id2label: Dict, is_bert: bool = True)
```

## Submodules

```
langml.baselines.cli
```

## Module Contents

### Functions

<i>baseline()</i>	LangML Baseline client
-------------------	------------------------

```
langml.baselines.cli.baseline()
```

```
LangML Baseline client
```

## Package Contents

### Classes

<i>BaselineModel</i>	
<i>BaseDataLoader</i>	
<i>Parameters</i>	Hyper-Parameters

```
class langml.baselines.BaselineModel
```

```
    abstract build_model(self, *args, **kwargs)
```

```
class langml.baselines.BaseDataLoader
```

```
    abstract static load_data()
```

```
    abstract make_iter(self, random: bool = False)
```

```
    abstract __len__(self)
```

```
    __call__(self, random: bool = False)
```

```
class langml.baselines.Parameters(data: Optional[Dict] = None)
```

```
Hyper-Parameters
```

```
    _wrap(self, value: Any)
```

```
    add(self, name, value)
```

`langml.common`

## Subpackages

`langml.common.evaluator`

## Submodules

`langml.common.evaluator.spearman`

## Module Contents

### Classes

---

*SpearmanEvaluator*

---

```
class langml.common.evaluator.spearman.SpearmanEvaluator(encoder: langml.tensor_typing.Models,
                                                         tokenizer: langml.tokenizer.Tokenizer)
```

```
    compute_corrcoef(self, data: List[Tuple[str, str, int]]) → float
```

## Package Contents

### Classes

---

*SpearmanEvaluator*

---

```
class langml.common.evaluator.SpearmanEvaluator(encoder: langml.tensor_typing.Models, tokenizer:
                                                langml.tokenizer.Tokenizer)
```

```
    compute_corrcoef(self, data: List[Tuple[str, str, int]]) → float
```

`langml.layers`

## Submodules

`langml.layers.attention`

## Module Contents

## Classes

---

*SelfAttention*

---

---

*SelfAdditiveAttention*

---

---

*ScaledDotProductAttention*

---

ScaledDotProductAttention

---

*MultiHeadAttention*

---

MultiHeadAttention

---

*GatedAttentionUnit*

---

Gated Attention Unit

---

```

class langml.layers.attention.SelfAttention(attention_units: Optional[int] = None, return_attention:
    bool = False, is_residual: bool = False,
    attention_activation: langml.tensor_typing.Activation =
    'relu', attention_epsilon: float = 10000000000.0,
    kernel_initializer: langml.tensor_typing.Initializer =
    'glorot_normal', kernel_regularizer:
    Optional[langml.tensor_typing.Regularizer] = None,
    kernel_constraint:
    Optional[langml.tensor_typing.Constraint] = None,
    bias_initializer: langml.tensor_typing.Initializer = 'zeros',
    bias_regularizer:
    Optional[langml.tensor_typing.Regularizer] = None,
    bias_constraint:
    Optional[langml.tensor_typing.Constraint] = None,
    use_attention_bias: bool = True, attention_penalty_weight:
    float = 0.0, **kwargs)

```

Bases: tensorflow.keras.layers.Layer

**get\_config**(*self*) → dict**build**(*self, input\_shape: langml.tensor\_typing.Tensors*)**call**(*self, inputs: langml.tensor\_typing.Tensors, mask: Optional[langml.tensor\_typing.Tensors] = None,
 \*\*kwargs*) → Union[List[langml.tensor\_typing.Tensors], langml.tensor\_typing.Tensors]**compute\_mask**(*self, inputs: langml.tensor\_typing.Tensors, mask: Optional[langml.tensor\_typing.Tensors] =
 None*) → Union[List[Union[langml.tensor\_typing.Tensors, None]],
 langml.tensor\_typing.Tensors]**\_attention\_penalty**(*self, attention: langml.tensor\_typing.Tensors*) → langml.tensor\_typing.Tensors**static get\_custom\_objects**() → dict**compute\_output\_shape**(*self, input\_shape: langml.tensor\_typing.Tensors*) →
 Union[List[langml.tensor\_typing.Tensors], langml.tensor\_typing.Tensors]

```

class langml.layers.attention.SelfAdditiveAttention(attention_units: Optional[int] = None,
                                                    return_attention: bool = False, is_residual: bool
                                                    = False, attention_activation:
                                                    langml.tensor_typing.Activation = 'relu',
                                                    attention_epsilon: float = 10000000000.0,
                                                    kernel_initializer:
                                                    langml.tensor_typing.Initializer =
                                                    'glorot_normal', kernel_regularizer:
                                                    Optional[langml.tensor_typing.Regularizer] =
                                                    None, kernel_constraint:
                                                    Optional[langml.tensor_typing.Constraint] =
                                                    None, bias_initializer:
                                                    langml.tensor_typing.Initializer = 'zeros',
                                                    bias_regularizer:
                                                    Optional[langml.tensor_typing.Regularizer] =
                                                    None, bias_constraint:
                                                    Optional[langml.tensor_typing.Constraint] =
                                                    None, use_attention_bias: bool = True,
                                                    attention_penalty_weight: float = 0.0,
                                                    **kwargs)

```

Bases: tensorflow.keras.layers.Layer

**get\_config**(self) → dict

**build**(self, input\_shape: langml.tensor\_typing.Tensors)

**call**(self, inputs: langml.tensor\_typing.Tensors, mask: Optional[langml.tensor\_typing.Tensors] = None,
 \*\*kwargs) → Union[List[langml.tensor\_typing.Tensors], langml.tensor\_typing.Tensors]

**compute\_mask**(self, inputs: langml.tensor\_typing.Tensors, mask: Optional[langml.tensor\_typing.Tensors] =
 None) → Union[List[Union[langml.tensor\_typing.Tensors, None]],
 langml.tensor\_typing.Tensors]

**\_attention\_penalty**(self, attention: langml.tensor\_typing.Tensors) → langml.tensor\_typing.Tensors

**static get\_custom\_objects**() → dict

**compute\_output\_shape**(self, input\_shape: langml.tensor\_typing.Tensors) →
 Union[List[langml.tensor\_typing.Tensors], langml.tensor\_typing.Tensors]

```

class langml.layers.attention.ScaledDotProductAttention(return_attention: bool = False,
                                                         history_only: bool = False, **kwargs)

```

Bases: tensorflow.keras.layers.Layer

ScaledDotProductAttention

\$Attention(Q, K, V) = \text{softmax}(\text{frac}\{Q K^T\} \{\text{sqrt}\{d\_k\}\}) V\$

<https://arxiv.org/pdf/1706.03762.pdf>

**get\_config**(self) → dict

**call**(self, inputs: langml.tensor\_typing.Tensors, mask: Optional[Union[langml.tensor\_typing.Tensors,
 List[langml.tensor\_typing.Tensors]]] = None, \*\*kwargs) → Union[List[langml.tensor\_typing.Tensors],
 langml.tensor\_typing.Tensors]

```
compute_mask(self, inputs: langml.tensor_typing.Tensors, mask:
    Optional[Union[langml.tensor_typing.Tensors, List[langml.tensor_typing.Tensors]]] =
    None) → Union[List[Union[langml.tensor_typing.Tensors, None]],
    langml.tensor_typing.Tensors]
```

```
static get_custom_objects() → dict
```

```
compute_output_shape(self, input_shape: Union[langml.tensor_typing.Tensors,
    List[langml.tensor_typing.Tensors]]) →
    Union[List[langml.tensor_typing.Tensors], langml.tensor_typing.Tensors]
```

```
class langml.layers.attention.MultiHeadAttention(head_num: int, return_attention: bool = False,
    attention_activation:
        langml.tensor_typing.Activation = 'relu',
    kernel_initializer: langml.tensor_typing.Initializer =
        'glorot_normal', kernel_regularizer:
        Optional[langml.tensor_typing.Regularizer] =
        None, kernel_constraint:
        Optional[langml.tensor_typing.Constraint] = None,
    bias_initializer: langml.tensor_typing.Initializer =
        'zeros', bias_regularizer:
        Optional[langml.tensor_typing.Regularizer] =
        None, bias_constraint:
        Optional[langml.tensor_typing.Constraint] = None,
    use_attention_bias: bool = True, history_only: bool
    = False, **kwargs)
```

Bases: tensorflow.keras.layers.Layer

MultiHeadAttention <https://arxiv.org/pdf/1706.03762.pdf>

```
get_config(self) → dict
```

```
build(self, input_shape: langml.tensor_typing.Tensors)
```

```
static _reshape_to_batches(x, head_num)
```

```
static _reshape_attention_from_batches(x, head_num)
```

```
static _reshape_from_batches(x, head_num)
```

```
static _reshape_mask(mask, head_num)
```

```
call(self, inputs: langml.tensor_typing.Tensors, mask: Optional[langml.tensor_typing.Tensors] = None,
    **kwargs) → langml.tensor_typing.Tensors
```

```
static get_custom_objects() → dict
```

```
compute_mask(self, inputs: langml.tensor_typing.Tensors, mask: Optional[langml.tensor_typing.Tensors] =
    None) → Union[List[Union[langml.tensor_typing.Tensors, None]],
    langml.tensor_typing.Tensors]
```

```
compute_output_shape(self, input_shape: Union[langml.tensor_typing.Tensors,
    List[langml.tensor_typing.Tensors]]) →
    Union[List[langml.tensor_typing.Tensors], langml.tensor_typing.Tensors]
```



```

class langml.layers.attention.GatedAttentionUnit(attention_units: int, attention_activation:
    langml.tensor_typing.Activation = 'relu',
    attention_normalizer:
    langml.tensor_typing.Activation = relu2,
    attention_epsilon: float = 10000000000.0,
    kernel_initializer: langml.tensor_typing.Initializer =
    'glorot_normal', kernel_regularizer:
    Optional[langml.tensor_typing.Regularizer] =
    None, kernel_constraint:
    Optional[langml.tensor_typing.Constraint] = None,
    bias_initializer: langml.tensor_typing.Initializer =
    'zeros', bias_regularizer:
    Optional[langml.tensor_typing.Regularizer] =
    None, bias_constraint:
    Optional[langml.tensor_typing.Constraint] = None,
    use_attention_bias: bool = True,
    use_attention_scale: bool = True,
    use_relative_position: bool = True, use_offset: bool
    = True, use_scale: bool = True, is_residual: bool =
    True, **kwargs)

Bases: tensorflow.keras.layers.Layer

Gated Attention Unit https://arxiv.org/abs/2202.10447

get_config(self) → dict

build(self, input_shape: langml.tensor_typing.Tensors)

apply_rotary_position_embeddings(self, sinusoidal: langml.tensor_typing.Tensors, *tensors)
    apply RoPE modified from: https://github.com/bojone/bert4keras/blob/master/bert4keras/backend.py#L310

attn(self, x: langml.tensor_typing.Tensors, v: langml.tensor_typing.Tensors, mask:
    Optional[langml.tensor_typing.Tensors] = None) → langml.tensor_typing.Tensors

call(self, inputs: langml.tensor_typing.Tensors, mask: Optional[langml.tensor_typing.Tensors] = None,
    **kwargs) → langml.tensor_typing.Tensors

compute_mask(self, inputs: langml.tensor_typing.Tensors, mask: Optional[langml.tensor_typing.Tensors] =
    None) → langml.tensor_typing.Tensors

compute_output_shape(self, input_shape: langml.tensor_typing.Tensors) → langml.tensor_typing.Tensors

static get_custom_objects() → dict

```

`langml.layers.crf`

## Module Contents

### Classes

---

[CRF](#)

---

```
class langml.layers.crf.CRF(output_dim: int, sparse_target: bool = True, **kwargs)
    Bases: tensorflow.keras.layers.Layer
    build(self, input_shape: langml.tensor_typing.Tensors)

    compute_mask(self, inputs: langml.tensor_typing.Tensors, mask: Optional[langml.tensor_typing.Tensors] =
        None)

    call(self, inputs: langml.tensor_typing.Tensors, sequence_lengths: Optional[langml.tensor_typing.Tensors]
        = None, training: Optional[Union[bool, int]] = None, mask: Optional[langml.tensor_typing.Tensors] =
        None, **kwargs) → langml.tensor_typing.Tensors

    property loss(self) → Callable
    property accuracy(self) → Callable

    compute_output_shape(self, input_shape: langml.tensor_typing.Tensors) → langml.tensor_typing.Tensors

    property trans(self) → langml.tensor_typing.Tensors
        transition parameters
    get_config(self) → dict
    static get_custom_objects() → dict
```

`langml.layers.layer_norm`

## Module Contents

### Classes

---

[\*LayerNorm\*](#)

---

```
class langml.layers.layer_norm.LayerNorm(center: bool = True, scale: bool = True, epsilon: float = 1e-07,
    gamma_initializer: langml.tensor_typing.Initializer = 'ones',
    gamma_regularizer:
    Optional[langml.tensor_typing.Regularizer] = None,
    gamma_constraint: Optional[langml.tensor_typing.Constraint]
    = None, beta_initializer: langml.tensor_typing.Initializer =
    'zeros', beta_regularizer:
    Optional[langml.tensor_typing.Regularizer] = None,
    beta_constraint: Optional[langml.tensor_typing.Constraint] =
    None, **kwargs)

    Bases: tensorflow.keras.layers.Layer
    get_config(self) → dict
    build(self, input_shape: langml.tensor_typing.Tensors)
    call(self, inputs: langml.tensor_typing.Tensors, **kwargs) → langml.tensor_typing.Tensors
```

**compute\_mask**(self, inputs: langml.tensor\_typing.Tensors, mask: Optional[langml.tensor\_typing.Tensors] = None) → Union[langml.tensor\_typing.Tensors, None]

**static get\_custom\_objects**() → dict

**compute\_output\_shape**(self, input\_shape: langml.tensor\_typing.Tensors) → langml.tensor\_typing.Tensors

## langml.layers.layers

### Module Contents

#### Classes

<i>AbsolutePositionEmbedding</i>	
<i>SineCosinePositionEmbedding</i>	Sine Cosine Position Embedding.
<i>ScaleOffset</i>	Scale Offset
<i>ConditionalLayerNormalization</i>	Conditional Layer Normalization

**class** langml.layers.layers.**AbsolutePositionEmbedding**(input\_dim: int, output\_dim: int, mode: str = 'add', embeddings\_initializer: langml.tensor\_typing.Initializer = 'uniform', embeddings\_regularizer: Optional[langml.tensor\_typing.Regularizer] = None, embeddings\_constraint: Optional[langml.tensor\_typing.Constraint] = None, mask\_zero: bool = False, \*\*kwargs)

Bases: langml.L.Layer

**get\_config**(self) → dict

**static get\_custom\_objects**() → dict

**build**(self, input\_shape: langml.tensor\_typing.Tensors)

**compute\_mask**(self, inputs: langml.tensor\_typing.Tensors, mask: Optional[langml.tensor\_typing.Tensors] = None) → langml.tensor\_typing.Tensors

**compute\_output\_shape**(self, input\_shape: langml.tensor\_typing.Tensors) → langml.tensor\_typing.Tensors

**call**(self, inputs: langml.tensor\_typing.Tensors, \*\*kwargs) → langml.tensor\_typing.Tensors

**class** langml.layers.layers.**SineCosinePositionEmbedding**(mode: str = 'add', output\_dim: Optional[int] = None, \*\*kwargs)

Bases: langml.L.Layer

Sine Cosine Position Embedding. <https://arxiv.org/pdf/1706.03762>

**get\_config**(self)

**static get\_custom\_objects**() → dict

**compute\_mask**(self, inputs: langml.tensor\_typing.Tensors, mask: Optional[langml.tensor\_typing.Tensors] = None) → langml.tensor\_typing.Tensors

**compute\_output\_shape**(self, input\_shape: langml.tensor\_typing.Tensors) → langml.tensor\_typing.Tensors

**call**(self, inputs: langml.tensor\_typing.Tensors, mask: Optional[langml.tensor\_typing.Tensors] = None, \*\*kwargs) → langml.tensor\_typing.Tensors

**class** langml.layers.layers.**ScaleOffset**(scale: bool = True, offset: bool = True, \*\*kwargs)

Bases: langml.L.Layer

Scale Offset

**get\_config**(self)

**build**(self, input\_shape: langml.tensor\_typing.Tensors)

**compute\_mask**(self, inputs: langml.tensor\_typing.Tensors, mask: Optional[langml.tensor\_typing.Tensors] = None)

**call**(self, inputs: langml.tensor\_typing.Tensors) → langml.tensor\_typing.Tensors

**compute\_output\_shape**(self, input\_shape: langml.tensor\_typing.Tensors) → langml.tensor\_typing.Tensors

**static get\_custom\_objects**() → dict

**class** langml.layers.layers.**ConditionalLayerNormalization**(center: bool = True, epsilon: Optional[float] = None, scale: bool = True, offset: bool = True, \*\*kwargs)

Bases: langml.L.Layer

Conditional Layer Normalization <https://arxiv.org/abs/2108.00449>

**get\_config**(self)

**build**(self, input\_shapes: langml.tensor\_typing.Tensors)

**compute\_mask**(self, inputs: langml.tensor\_typing.Tensors, mask: Optional[langml.tensor\_typing.Tensors] = None)

**call**(self, inputs: List[langml.tensor\_typing.Tensors]) → langml.tensor\_typing.Tensors

**compute\_output\_shape**(self, input\_shape: langml.tensor\_typing.Tensors) → langml.tensor\_typing.Tensors

**static get\_custom\_objects**() → dict

## Package Contents

## Classes

<i>CRF</i>	
<i>LayerNorm</i>	
<i>AbsolutePositionEmbedding</i>	
<i>SineCosinePositionEmbedding</i>	Sine Cosine Position Embedding.
<i>ScaleOffset</i>	Scale Offset
<i>ConditionalLayerNormalization</i>	Conditional Layer Normalization
<i>SelfAttention</i>	
<i>SelfAdditiveAttention</i>	
<i>ScaledDotProductAttention</i>	ScaledDotProductAttention
<i>MultiHeadAttention</i>	MultiHeadAttention
<i>GatedAttentionUnit</i>	Gated Attention Unit

## Attributes

<i>TF_KERAS</i>
<i>custom_objects</i>

langml.layers.TF\_KERAS

**class** langml.layers.CRF(output\_dim: int, sparse\_target: bool = True, \*\*kwargs)

Bases: tensorflow.keras.layers.Layer

**build**(self, input\_shape: langml.tensor\_typing.Tensors)

**compute\_mask**(self, inputs: langml.tensor\_typing.Tensors, mask: Optional[langml.tensor\_typing.Tensors] = None)

**call**(self, inputs: langml.tensor\_typing.Tensors, sequence\_lengths: Optional[langml.tensor\_typing.Tensors] = None, training: Optional[Union[bool, int]] = None, mask: Optional[langml.tensor\_typing.Tensors] = None, \*\*kwargs) → langml.tensor\_typing.Tensors

**property** loss(self) → Callable

**property** accuracy(self) → Callable

**compute\_output\_shape**(self, input\_shape: langml.tensor\_typing.Tensors) → langml.tensor\_typing.Tensors

**property** trans(self) → langml.tensor\_typing.Tensors  
transition parameters

**get\_config**(self) → dict

**static** get\_custom\_objects() → dict

```
class langml.layers.LayerNorm(center: bool = True, scale: bool = True, epsilon: float = 1e-07,  
                             gamma_initializer: langml.tensor_typing.Initializer = 'ones',  
                             gamma_regularizer: Optional[langml.tensor_typing.Regularizer] = None,  
                             gamma_constraint: Optional[langml.tensor_typing.Constraint] = None,  
                             beta_initializer: langml.tensor_typing.Initializer = 'zeros', beta_regularizer:  
                             Optional[langml.tensor_typing.Regularizer] = None, beta_constraint:  
                             Optional[langml.tensor_typing.Constraint] = None, **kwargs)  
  
    Bases: tensorflow.keras.layers.Layer  
  
    get_config(self) → dict  
  
    build(self, input_shape: langml.tensor_typing.Tensors)  
  
    call(self, inputs: langml.tensor_typing.Tensors, **kwargs) → langml.tensor_typing.Tensors  
  
    compute_mask(self, inputs: langml.tensor_typing.Tensors, mask: Optional[langml.tensor_typing.Tensors] =  
                  None) → Union[langml.tensor_typing.Tensors, None]  
  
    static get_custom_objects() → dict  
  
    compute_output_shape(self, input_shape: langml.tensor_typing.Tensors) → langml.tensor_typing.Tensors  
  
class langml.layers.AbsolutePositionEmbedding(input_dim: int, output_dim: int, mode: str = 'add',  
                                              embeddings_initializer: langml.tensor_typing.Initializer  
                                              = 'uniform', embeddings_regularizer:  
                                              Optional[langml.tensor_typing.Regularizer] = None,  
                                              embeddings_constraint:  
                                              Optional[langml.tensor_typing.Constraint] = None,  
                                              mask_zero: bool = False, **kwargs)  
  
    Bases: langml.L.Layer  
  
    get_config(self) → dict  
  
    static get_custom_objects() → dict  
  
    build(self, input_shape: langml.tensor_typing.Tensors)  
  
    compute_mask(self, inputs: langml.tensor_typing.Tensors, mask: Optional[langml.tensor_typing.Tensors] =  
                  None) → langml.tensor_typing.Tensors  
  
    compute_output_shape(self, input_shape: langml.tensor_typing.Tensors) → langml.tensor_typing.Tensors  
  
    call(self, inputs: langml.tensor_typing.Tensors, **kwargs) → langml.tensor_typing.Tensors  
  
class langml.layers.SineCosinePositionEmbedding(mode: str = 'add', output_dim: Optional[int] = None,  
                                              **kwargs)  
  
    Bases: langml.L.Layer  
  
    Sine Cosine Position Embedding. https://arxiv.org/pdf/1706.03762  
  
    get_config(self)  
  
    static get_custom_objects() → dict  
  
    compute_mask(self, inputs: langml.tensor_typing.Tensors, mask: Optional[langml.tensor_typing.Tensors] =  
                  None) → langml.tensor_typing.Tensors  
  
    compute_output_shape(self, input_shape: langml.tensor_typing.Tensors) → langml.tensor_typing.Tensors
```

```

    call(self, inputs: langml.tensor_typing.Tensors, mask: Optional[langml.tensor_typing.Tensors] = None,
          **kwargs) → langml.tensor_typing.Tensors

class langml.layers.ScaleOffset(scale: bool = True, offset: bool = True, **kwargs)
    Bases: langml.L.Layer
    Scale Offset
    get_config(self)

    build(self, input_shape: langml.tensor_typing.Tensors)

    compute_mask(self, inputs: langml.tensor_typing.Tensors, mask: Optional[langml.tensor_typing.Tensors] =
                  None)

    call(self, inputs: langml.tensor_typing.Tensors) → langml.tensor_typing.Tensors

    compute_output_shape(self, input_shape: langml.tensor_typing.Tensors) → langml.tensor_typing.Tensors

    static get_custom_objects() → dict

class langml.layers.ConditionalLayerNormalization(center: bool = True, epsilon: Optional[float] =
                                                  None, scale: bool = True, offset: bool = True,
                                                  **kwargs)

    Bases: langml.L.Layer
    Conditional Layer Normalization https://arxiv.org/abs/2108.00449
    get_config(self)

    build(self, input_shapes: langml.tensor_typing.Tensors)

    compute_mask(self, inputs: langml.tensor_typing.Tensors, mask: Optional[langml.tensor_typing.Tensors] =
                  None)

    call(self, inputs: List[langml.tensor_typing.Tensors]) → langml.tensor_typing.Tensors

    compute_output_shape(self, input_shape: langml.tensor_typing.Tensors) → langml.tensor_typing.Tensors

    static get_custom_objects() → dict

class langml.layers.SelfAttention(attention_units: Optional[int] = None, return_attention: bool = False,
                                  is_residual: bool = False, attention_activation:
                                  langml.tensor_typing.Activation = 'relu', attention_epsilon: float =
                                  10000000000.0, kernel_initializer: langml.tensor_typing.Initializer =
                                  'glorot_normal', kernel_regularizer:
                                  Optional[langml.tensor_typing.Regularizer] = None, kernel_constraint:
                                  Optional[langml.tensor_typing.Constraint] = None, bias_initializer:
                                  langml.tensor_typing.Initializer = 'zeros', bias_regularizer:
                                  Optional[langml.tensor_typing.Regularizer] = None, bias_constraint:
                                  Optional[langml.tensor_typing.Constraint] = None, use_attention_bias:
                                  bool = True, attention_penalty_weight: float = 0.0, **kwargs)

    Bases: tensorflow.keras.layers.Layer
    get_config(self) → dict

    build(self, input_shape: langml.tensor_typing.Tensors)

```

```
call(self, inputs: langml.tensor_typing.Tensors, mask: Optional[langml.tensor_typing.Tensors] = None,
      **kwargs) → Union[List[langml.tensor_typing.Tensors], langml.tensor_typing.Tensors]

compute_mask(self, inputs: langml.tensor_typing.Tensors, mask: Optional[langml.tensor_typing.Tensors] =
               None) → Union[List[Union[langml.tensor_typing.Tensors, None]],
                             langml.tensor_typing.Tensors]

_attention_penalty(self, attention: langml.tensor_typing.Tensors) → langml.tensor_typing.Tensors

static get_custom_objects() → dict

compute_output_shape(self, input_shape: langml.tensor_typing.Tensors) →
                       Union[List[langml.tensor_typing.Tensors], langml.tensor_typing.Tensors]

class langml.layers.SelfAdditiveAttention(attention_units: Optional[int] = None, return_attention: bool
                                          = False, is_residual: bool = False, attention_activation:
                                          langml.tensor_typing.Activation = 'relu', attention_epsilon:
                                          float = 10000000000.0, kernel_initializer:
                                          langml.tensor_typing.Initializer = 'glorot_normal',
                                          kernel_regularizer:
                                          Optional[langml.tensor_typing.Regularizer] = None,
                                          kernel_constraint:
                                          Optional[langml.tensor_typing.Constraint] = None,
                                          bias_initializer: langml.tensor_typing.Initializer = 'zeros',
                                          bias_regularizer:
                                          Optional[langml.tensor_typing.Regularizer] = None,
                                          bias_constraint: Optional[langml.tensor_typing.Constraint]
                                          = None, use_attention_bias: bool = True,
                                          attention_penalty_weight: float = 0.0, **kwargs)

Bases: tensorflow.keras.layers.Layer

get_config(self) → dict

build(self, input_shape: langml.tensor_typing.Tensors)

call(self, inputs: langml.tensor_typing.Tensors, mask: Optional[langml.tensor_typing.Tensors] = None,
      **kwargs) → Union[List[langml.tensor_typing.Tensors], langml.tensor_typing.Tensors]

compute_mask(self, inputs: langml.tensor_typing.Tensors, mask: Optional[langml.tensor_typing.Tensors] =
               None) → Union[List[Union[langml.tensor_typing.Tensors, None]],
                             langml.tensor_typing.Tensors]

_attention_penalty(self, attention: langml.tensor_typing.Tensors) → langml.tensor_typing.Tensors

static get_custom_objects() → dict

compute_output_shape(self, input_shape: langml.tensor_typing.Tensors) →
                       Union[List[langml.tensor_typing.Tensors], langml.tensor_typing.Tensors]

class langml.layers.ScaledDotProductAttention(return_attention: bool = False, history_only: bool =
                                          False, **kwargs)
```

Bases: tensorflow.keras.layers.Layer

ScaledDotProductAttention

$\text{\$Attention}(Q, K, V) = \text{softmax}(\text{frac}\{Q K^T\} \{\text{sqrt}\{d_k\}\}) V$

<https://arxiv.org/pdf/1706.03762.pdf>



```

get_config(self) → dict

call(self, inputs: langml.tensor_typing.Tensors, mask: Optional[Union[langml.tensor_typing.Tensors,
    List[langml.tensor_typing.Tensors]]] = None, **kwargs) → Union[List[langml.tensor_typing.Tensors],
    langml.tensor_typing.Tensors]

compute_mask(self, inputs: langml.tensor_typing.Tensors, mask:
    Optional[Union[langml.tensor_typing.Tensors, List[langml.tensor_typing.Tensors]]] =
    None) → Union[List[Union[langml.tensor_typing.Tensors, None]],
    langml.tensor_typing.Tensors]

static get_custom_objects() → dict

compute_output_shape(self, input_shape: Union[langml.tensor_typing.Tensors,
    List[langml.tensor_typing.Tensors]]) →
    Union[List[langml.tensor_typing.Tensors], langml.tensor_typing.Tensors]

class langml.layers.MultiHeadAttention(head_num: int, return_attention: bool = False,
    attention_activation: langml.tensor_typing.Activation = 'relu',
    kernel_initializer: langml.tensor_typing.Initializer =
    'glorot_normal', kernel_regularizer:
    Optional[langml.tensor_typing.Regularizer] = None,
    kernel_constraint: Optional[langml.tensor_typing.Constraint] =
    None, bias_initializer: langml.tensor_typing.Initializer = 'zeros',
    bias_regularizer: Optional[langml.tensor_typing.Regularizer] =
    None, bias_constraint:
    Optional[langml.tensor_typing.Constraint] = None,
    use_attention_bias: bool = True, history_only: bool = False,
    **kwargs)

Bases: tensorflow.keras.layers.Layer

MultiHeadAttention https://arxiv.org/pdf/1706.03762.pdf

get_config(self) → dict

build(self, input_shape: langml.tensor_typing.Tensors)

static _reshape_to_batches(x, head_num)

static _reshape_attention_from_batches(x, head_num)

static _reshape_from_batches(x, head_num)

static _reshape_mask(mask, head_num)

call(self, inputs: langml.tensor_typing.Tensors, mask: Optional[langml.tensor_typing.Tensors] = None,
    **kwargs) → langml.tensor_typing.Tensors

static get_custom_objects() → dict

compute_mask(self, inputs: langml.tensor_typing.Tensors, mask: Optional[langml.tensor_typing.Tensors] =
    None) → Union[List[Union[langml.tensor_typing.Tensors, None]],
    langml.tensor_typing.Tensors]

compute_output_shape(self, input_shape: Union[langml.tensor_typing.Tensors,
    List[langml.tensor_typing.Tensors]]) →
    Union[List[langml.tensor_typing.Tensors], langml.tensor_typing.Tensors]

```

```
class langml.layers.GatedAttentionUnit(attention_units: int, attention_activation:
    langml.tensor_typing.Activation = 'relu', attention_normalizer:
    langml.tensor_typing.Activation = relu2, attention_epsilon: float
    = 10000000000.0, kernel_initializer:
    langml.tensor_typing.Initializer = 'glorot_normal',
    kernel_regularizer: Optional[langml.tensor_typing.Regularizer]
    = None, kernel_constraint:
    Optional[langml.tensor_typing.Constraint] = None,
    bias_initializer: langml.tensor_typing.Initializer = 'zeros',
    bias_regularizer: Optional[langml.tensor_typing.Regularizer] =
    None, bias_constraint:
    Optional[langml.tensor_typing.Constraint] = None,
    use_attention_bias: bool = True, use_attention_scale: bool =
    True, use_relative_position: bool = True, use_offset: bool = True,
    use_scale: bool = True, is_residual: bool = True, **kwargs)
```

Bases: tensorflow.keras.layers.Layer

Gated Attention Unit <https://arxiv.org/abs/2202.10447>

**get\_config**(self) → dict

**build**(self, input\_shape: langml.tensor\_typing.Tensors)

**apply\_rotary\_position\_embeddings**(self, sinusoidal: langml.tensor\_typing.Tensors, \*tensors)

apply RoPE modified from: <https://github.com/bojone/bert4keras/blob/master/bert4keras/backend.py#L310>

**attn**(self, x: langml.tensor\_typing.Tensors, v: langml.tensor\_typing.Tensors, mask:
 Optional[langml.tensor\_typing.Tensors] = None) → langml.tensor\_typing.Tensors

**call**(self, inputs: langml.tensor\_typing.Tensors, mask: Optional[langml.tensor\_typing.Tensors] = None,
 \*\*kwargs) → langml.tensor\_typing.Tensors

**compute\_mask**(self, inputs: langml.tensor\_typing.Tensors, mask: Optional[langml.tensor\_typing.Tensors] =
 None) → langml.tensor\_typing.Tensors

**compute\_output\_shape**(self, input\_shape: langml.tensor\_typing.Tensors) → langml.tensor\_typing.Tensors

**static get\_custom\_objects**() → dict

langml.layers.custom\_objects

langml.plm

Submodules

langml.plm.albert

Module Contents

## Functions

---

`load_albert`(`config_path`: str, `checkpoint_path`: str, Load pretrained ALBERT  
`seq_len`: Optional[int] = None, `pretraining`: bool =  
False, `with_mlm`: bool = True, `with_nsp`: bool = True,  
`lazy_restore`: bool = False, `weight_prefix`: Optional[str]  
= None, `dropout_rate`: float = 0.0, `**kwargs`) →  
Union[Tuple[langml.tensor\_typing.Models, Callable],  
Tuple[langml.tensor\_typing.Models, Callable,  
Callable]]

---

`langml.plm.albert.load_albert`(`config_path`: str, `checkpoint_path`: str, `seq_len`: Optional[int] = None,  
`pretraining`: bool = False, `with_mlm`: bool = True, `with_nsp`: bool = True,  
`lazy_restore`: bool = False, `weight_prefix`: Optional[str] = None,  
`dropout_rate`: float = 0.0, `**kwargs`) →  
Union[Tuple[langml.tensor\_typing.Models, Callable],  
Tuple[langml.tensor\_typing.Models, Callable, Callable]]

Load pretrained ALBERT :param - `config_path`: str, path of albert config :param - `checkpoint_path`: str, path  
of albert checkpoint :param - `seq_len`: Optional[int], specify fixed input sequence length, default None :param  
- `pretraining`: bool, pretraining mode, default False :param - `with_mlm`: bool, whether to use mlm task in pre-  
training, default True :param - `with_nsp`: bool, whether to use nsp/sop task in pretraining, default True :param -  
`lazy_restore`: bool, whether to restore pretrained weights lazily, default False.

Set it as True for distributed training.

### Parameters

**`weight_prefix`** (-) –

**Optional[str], prefix name of weights, default None.**

You can set a prefix name in unshared siamese networks.

- `dropout_rate`: float, dropout rate, default 0.

### Returns

keras model - bert: bert instance - restore: conditionally, it will return when `lazy_restore=True`

### Return type

- model

`langml.plm.bert`

## Module Contents

### Classes

---

*BERT*

---

## Functions

---

```
load_bert(config_path: str, checkpoint_path: str, Load pretrained BERT/RoBERTa
seq_len: Optional[int] = None, pretraining: bool =
False, with_mlm: bool = True, with_nsp: bool = True,
lazy_restore: bool = False, weight_prefix: Optional[str]
= None, dropout_rate: float = 0.0, **kwargs) →
Union[Tuple[langml.tensor_typing.Models, Callable],
Tuple[langml.tensor_typing.Models, Callable,
Callable]]
```

---

```
class langml.plm.bert.BERT(vocab_size: int, position_size: int = 512, seq_len: int = 512, embedding_dim: int
= 768, hidden_dim: Optional[int] = None, transformer_blocks: int = 12,
attention_heads: int = 12, intermediate_size: int = 3072, dropout_rate: float =
0.1, attention_activation: langml.tensor_typing.Activation = None,
feed_forward_activation: langml.tensor_typing.Activation = 'gelu',
initializer_range: float = 0.02, pretraining: bool = False, trainable_prefixs:
Optional[List] = None, share_weights: bool = False, weight_prefix:
Optional[str] = None)
```

```
get_weight_name(self, name: str) → str
```

```
build(self)
```

```
get_inputs(self) → List[langml.tensor_typing.Tensors]
```

```
get_embedding(self, inputs: List[langml.tensor_typing.Tensors]) → List[langml.tensor_typing.Tensors]
```

```
is_trainable(self, layer: tensorflow.keras.layers.Layer) → bool
```

```
__call__(self, inputs: Optional[Union[Tuple, List]] = None, return_model: bool = True, with_mlm: bool =
True, with_nsp: bool = True, custom_embedding_callback: Optional[Callable] = None) →
langml.tensor_typing.Models
```

```
langml.plm.bert.load_bert(config_path: str, checkpoint_path: str, seq_len: Optional[int] = None, pretraining:
bool = False, with_mlm: bool = True, with_nsp: bool = True, lazy_restore: bool =
False, weight_prefix: Optional[str] = None, dropout_rate: float = 0.0, **kwargs)
→ Union[Tuple[langml.tensor_typing.Models, Callable],
Tuple[langml.tensor_typing.Models, Callable, Callable]]
```

Load pretrained BERT/RoBERTa :param - config\_path: str, path of albert config :param - checkpoint\_path: str, path of albert checkpoint :param - seq\_len: Optional[int], specify fixed input sequence length, default None :param - pretraining: bool, pretraining mode, default False :param - with\_mlm: bool, whether to use mlm task in pretraining, default True :param - with\_nsp: bool, whether to use nsp task in pretraining, default True :param - lazy\_restore: bool, whether to restore pretrained weights lazily, default False.

Set it as True for distributed training.

### Parameters

- **weight\_prefix** (-) – Optional[str], prefix name of weights, default None. You can set a prefix name in unshared siamese networks.
- **dropout\_rate** (-) – float, dropout rate, default 0.

### Returns

keras model - bert: bert instance - restore: conditionally, it will return when lazy\_restore=True

**Return type**

- model

`langml.plm.layers`**Module Contents****Classes***TokenEmbedding**EmbeddingMatching**Masked*

Generate output mask based on the given mask.

**class** `langml.plm.layers.TokenEmbedding`Bases: `tensorflow.keras.layers.Embedding`**static** `get_custom_objects()` → dict**compute\_mask**(*self*, *inputs*: `langml.tensor_typing.Tensors`, *mask*: `Optional[langml.tensor_typing.Tensors]` = `None`) → `List[Union[langml.tensor_typing.Tensors, None]]`**call**(*self*, *inputs*: `langml.tensor_typing.Tensors`) → `List[langml.tensor_typing.Tensors]`**compute\_output\_shape**(*self*, *input\_shape*: `langml.tensor_typing.Tensors`) → `List[langml.tensor_typing.Tensors]`

**class** `langml.plm.layers.EmbeddingMatching`(*initializer*: `langml.tensor_typing.Initializer` = 'zeros',  
*regularizer*: `Optional[langml.tensor_typing.Regularizer]` = `None`, *constraint*: `Optional[langml.tensor_typing.Constraint]` = `None`, *use\_bias*: `bool` = `True`, *use\_softmax*: `bool` = `True`,  
*\*\*kwargs*)

Bases: `tensorflow.keras.layers.Layer`**get\_config**(*self*) → dict**build**(*self*, *input\_shape*: `langml.tensor_typing.Tensors`)**compute\_mask**(*self*, *inputs*: `langml.tensor_typing.Tensors`, *mask*: `Optional[langml.tensor_typing.Tensors]` = `None`) → `langml.tensor_typing.Tensors`**call**(*self*, *inputs*: `langml.tensor_typing.Tensors`, *mask*: `Optional[langml.tensor_typing.Tensors]` = `None`,  
*\*\*kwargs*) → `langml.tensor_typing.Tensors`**static** `get_custom_objects()` → dict**compute\_output\_shape**(*self*, *input\_shape*: `langml.tensor_typing.Tensors`) → `langml.tensor_typing.Tensors`**class** `langml.plm.layers.Masked`(*return\_masked*: `bool` = `False`, *\*\*kwargs*)Bases: `tensorflow.keras.layers.Layer`Generate output mask based on the given mask. <https://arxiv.org/pdf/1810.04805.pdf>

**static** `get_custom_objects()` → dict

**get\_config**(*self*) → dict

**compute\_mask**(*self*, inputs: langml.tensor\_typing.Tensors, mask: Optional[langml.tensor\_typing.Tensors] = None) → Union[List[Union[langml.tensor\_typing.Tensors, None]], langml.tensor\_typing.Tensors]

**call**(*self*, inputs: langml.tensor\_typing.Tensors, mask: Optional[langml.tensor\_typing.Tensors] = None, \*\*kwargs) → langml.tensor\_typing.Tensors

**compute\_output\_shape**(*self*, input\_shape: langml.tensor\_typing.Tensors) → Union[List[langml.tensor\_typing.Tensors], langml.tensor\_typing.Tensors]

## Package Contents

### Classes

---

*TokenEmbedding*

---

*EmbeddingMatching*

---

*Masked*

Generate output mask based on the given mask.

---

### Functions

---

*load\_bert*(config\_path: str, checkpoint\_path: str, Load pretrained BERT/RoBERTa  
seq\_len: Optional[int] = None, pretraining: bool =  
False, with\_mlm: bool = True, with\_nsp: bool = True,  
lazy\_restore: bool = False, weight\_prefix: Optional[str]  
= None, dropout\_rate: float = 0.0, \*\*kwargs) →  
Union[Tuple[langml.tensor\_typing.Models, Callable],  
Tuple[langml.tensor\_typing.Models, Callable],  
Callable]]

---

*load\_albert*(config\_path: str, checkpoint\_path: str, Load pretrained ALBERT  
seq\_len: Optional[int] = None, pretraining: bool =  
False, with\_mlm: bool = True, with\_nsp: bool = True,  
lazy\_restore: bool = False, weight\_prefix: Optional[str]  
= None, dropout\_rate: float = 0.0, \*\*kwargs) →  
Union[Tuple[langml.tensor\_typing.Models, Callable],  
Tuple[langml.tensor\_typing.Models, Callable],  
Callable]]

---

## Attributes

---

### *custom\_objects*

---

#### **class** langml.plm.TokenEmbedding

Bases: tensorflow.keras.layers.Embedding

**static** get\_custom\_objects() → dict

**compute\_mask**(self, inputs: langml.tensor\_typing.Tensors, mask: Optional[langml.tensor\_typing.Tensors] = None) → List[Union[langml.tensor\_typing.Tensors, None]]

**call**(self, inputs: langml.tensor\_typing.Tensors) → List[langml.tensor\_typing.Tensors]

**compute\_output\_shape**(self, input\_shape: langml.tensor\_typing.Tensors) → List[langml.tensor\_typing.Tensors]

#### **class** langml.plm.EmbeddingMatching(initializer: langml.tensor\_typing.Initializer = 'zeros', regularizer: Optional[langml.tensor\_typing.Regularizer] = None, constraint: Optional[langml.tensor\_typing.Constraint] = None, use\_bias: bool = True, use\_softmax: bool = True, \*\*kwargs)

Bases: tensorflow.keras.layers.Layer

**get\_config**(self) → dict

**build**(self, input\_shape: langml.tensor\_typing.Tensors)

**compute\_mask**(self, inputs: langml.tensor\_typing.Tensors, mask: Optional[langml.tensor\_typing.Tensors] = None) → langml.tensor\_typing.Tensors

**call**(self, inputs: langml.tensor\_typing.Tensors, mask: Optional[langml.tensor\_typing.Tensors] = None, \*\*kwargs) → langml.tensor\_typing.Tensors

**static** get\_custom\_objects() → dict

**compute\_output\_shape**(self, input\_shape: langml.tensor\_typing.Tensors) → langml.tensor\_typing.Tensors

#### **class** langml.plm.Masked(return\_masked: bool = False, \*\*kwargs)

Bases: tensorflow.keras.layers.Layer

Generate output mask based on the given mask. <https://arxiv.org/pdf/1810.04805.pdf>

**static** get\_custom\_objects() → dict

**get\_config**(self) → dict

**compute\_mask**(self, inputs: langml.tensor\_typing.Tensors, mask: Optional[langml.tensor\_typing.Tensors] = None) → Union[List[Union[langml.tensor\_typing.Tensors, None]], langml.tensor\_typing.Tensors]

**call**(self, inputs: langml.tensor\_typing.Tensors, mask: Optional[langml.tensor\_typing.Tensors] = None, \*\*kwargs) → langml.tensor\_typing.Tensors

**compute\_output\_shape**(self, input\_shape: langml.tensor\_typing.Tensors) → Union[List[langml.tensor\_typing.Tensors], langml.tensor\_typing.Tensors]

```
langml.plm.load_bert(config_path: str, checkpoint_path: str, seq_len: Optional[int] = None, pretraining: bool  
= False, with_mlm: bool = True, with_nsp: bool = True, lazy_restore: bool = False,  
weight_prefix: Optional[str] = None, dropout_rate: float = 0.0, **kwargs) →  
Union[Tuple[langml.tensor_typing.Models, Callable],  
Tuple[langml.tensor_typing.Models, Callable, Callable]]
```

Load pretrained BERT/RoBERTa :param - config\_path: str, path of bert config :param - checkpoint\_path: str, path of bert checkpoint :param - seq\_len: Optional[int], specify fixed input sequence length, default None :param - pretraining: bool, pretraining mode, default False :param - with\_mlm: bool, whether to use mlm task in pretraining, default True :param - with\_nsp: bool, whether to use nsp task in pretraining, default True :param - lazy\_restore: bool, whether to restore pretrained weights lazily, default False.

Set it as True for distributed training.

#### Parameters

- **weight\_prefix** (-) – Optional[str], prefix name of weights, default None. You can set a prefix name in unshared siamese networks.
- **dropout\_rate** (-) – float, dropout rate, default 0.

#### Returns

keras model - bert: bert instance - restore: conditionally, it will return when lazy\_restore=True

#### Return type

- model

```
langml.plm.load_albert(config_path: str, checkpoint_path: str, seq_len: Optional[int] = None, pretraining:  
bool = False, with_mlm: bool = True, with_nsp: bool = True, lazy_restore: bool =  
False, weight_prefix: Optional[str] = None, dropout_rate: float = 0.0, **kwargs) →  
Union[Tuple[langml.tensor_typing.Models, Callable],  
Tuple[langml.tensor_typing.Models, Callable, Callable]]
```

Load pretrained ALBERT :param - config\_path: str, path of bert config :param - checkpoint\_path: str, path of bert checkpoint :param - seq\_len: Optional[int], specify fixed input sequence length, default None :param - pretraining: bool, pretraining mode, default False :param - with\_mlm: bool, whether to use mlm task in pretraining, default True :param - with\_nsp: bool, whether to use nsp/sop task in pretraining, default True :param - lazy\_restore: bool, whether to restore pretrained weights lazily, default False.

Set it as True for distributed training.

#### Parameters

**weight\_prefix** (-) –

**Optional[str], prefix name of weights, default None.**

You can set a prefix name in unshared siamese networks.

- dropout\_rate: float, dropout rate, default 0.

#### Returns

keras model - bert: bert instance - restore: conditionally, it will return when lazy\_restore=True

#### Return type

- model

`langml.plm.custom_objects`



`langml.prompt`

## Subpackages

`langml.prompt.clf`

## Submodules

`langml.prompt.clf.ptuning`

## Module Contents

## Classes

---

*DataGenerator*

---

---

*PTuningForClassification*

---

```
class langml.prompt.clf.ptuning.DataGenerator(data: List[str], labels: List[str], tokenizer:
                                             langml.tokenizer.Tokenizer, template:
                                             langml.prompt.base.Template, batch_size: int = 32)
```

Bases: *langml.prompt.base.BaseDataGenerator*

```
__len__(self)
```

```
make_iter(self, random: bool = False)
```

```
class langml.prompt.clf.ptuning.PTuningForClassification(prompt_model: BasePromptModel,
                                                         tokenizer: langml.tokenizer.Tokenizer)
```

Bases: *langml.prompt.base.BasePromptTask*

```
fit(self, data: List[str], labels: List[str], valid_data: Optional[List[str]] = None, valid_labels:
    Optional[List[str]] = None, model_path: Optional[str] = None, epoch: int = 20, batch_size: int = 16,
    early_stop: int = 10, do_shuffle: bool = True, f1_average: str = 'macro', verbose: int = 1)
```

Fitting ptuning model for classification :param - data: List[str], texts of training data :param - labels: List[Union[str, List[str]]], training labels :param - valid\_data: List[str], texts of valid data :param - valid\_labels: List[Union[str, List[str]]], labels of valid data :param - model\_path: Optional[str], path to save model, default *None*, do not to save model :param - epoch: int, epochs to train :param - batch\_size: int, batch size, :param - early\_stop: int, patience of early stop :param - do\_shuffle: whether to shuffle data in training phase :param - f1\_average: str, { 'micro', 'macro', 'samples', 'weighted', 'binary' } or *None* :param - verbose: int, 0 = silent, 1 = progress bar, 2 = one line per epoch

```
predict(self, text: str) → str
```

```
load(self, model_path: str)
```

load model :param - model\_path: str, model path

langml.prompt.clf.utils

## Module Contents

### Classes

---

*MetricsCallback*

---

### Functions

---

*merge\_template\_tokens*(template\_ids: List[int], token\_ids: List[int], max\_length: Optional[int] = None) → Tuple[List[int], List[int]]

---

Merge template and token ids

langml.prompt.clf.utils.**merge\_template\_tokens**(*template\_ids: List[int], token\_ids: List[int], max\_length: Optional[int] = None*) → Tuple[List[int], List[int]]

Merge template and token ids :param - template\_ids: List[int], template ids :param - token\_ids: List[int], token ids :param - max\_length: int, max length

#### Returns

List[int], merged token ids - template\_mask: List[int], template mask

#### Return type

- token\_ids

**class** langml.prompt.clf.utils.**MetricsCallback**(*data: List[str], labels: List[str], mask\_id: int, template: langml.prompt.base.Template, patience: int = 10, batch\_size: int = 32, model\_path: Optional[str] = None, fl\_average: str = 'macro'*)

Bases: langml.keras.callbacks.Callback

**on\_train\_begin**(*self, logs=None*)

**on\_epoch\_end**(*self, epoch, logs=None*)

**on\_train\_end**(*self, logs=None*)

## Package Contents

### Classes

---

*PTuningForClassification*

---

**class** langml.prompt.clf.**PTuningForClassification**(*prompt\_model: BasePromptModel, tokenizer: langml.tokenizer.Tokenizer*)

Bases: *langml.prompt.base.BasePromptTask*

**fit**(*self*, *data*: List[str], *labels*: List[str], *valid\_data*: Optional[List[str]] = None, *valid\_labels*: Optional[List[str]] = None, *model\_path*: Optional[str] = None, *epoch*: int = 20, *batch\_size*: int = 16, *early\_stop*: int = 10, *do\_shuffle*: bool = True, *f1\_average*: str = 'macro', *verbose*: int = 1)

Fitting ptuning model for classification :param - *data*: List[str], texts of training data :param - *labels*: List[Union[str, List[str]]], training labels :param - *valid\_data*: List[str], texts of valid data :param - *valid\_labels*: List[Union[str, List[str]]], labels of valid data :param - *model\_path*: Optional[str], path to save model, default *None*, do not to save model :param - *epoch*: int, epochs to train :param - *batch\_size*: int, batch size, :param - *early\_stop*: int, patience of early stop :param - *do\_shuffle*: whether to shuffle data in training phase :param - *f1\_average*: str, {'micro', 'macro', 'samples', 'weighted', 'binary'} or None :param - *verbose*: int, 0 = silent, 1 = progress bar, 2 = one line per epoch

**predict**(*self*, *text*: str) → str

**load**(*self*, *model\_path*: str)

load model :param - *model\_path*: str, model path

## langml.prompt.models

### Submodules

#### langml.prompt.models.ptuning

Implementation P-Tuning

Paper: GPT Understands, Too URL: <https://arxiv.org/pdf/2103.10385.pdf>

### Module Contents

#### Classes

---

*PartialEmbedding*

---

*PTuningPrompt*

---

```
class langml.prompt.models.ptuning.PartialEmbedding(input_dim: int, output_dim: int, active_start: int,
                                                    active_end: int, embeddings_initializer:
                                                    Optional[langml.tensor_typing.Initializer] =
                                                    'uniform', embeddings_regularizer:
                                                    Optional[langml.tensor_typing.Regularizer] =
                                                    None, activity_regularizer:
                                                    Optional[langml.tensor_typing.Regularizer] =
                                                    None, embeddings_constraint:
                                                    Optional[langml.tensor_typing.Constraint] =
                                                    None, mask_zero: bool = False, input_length:
                                                    Optional[int] = None, **kwargs)
```

Bases: langml.L.Embedding

**static** **get\_custom\_objects**() → dict

**compute\_mask**(self, inputs: langml.tensor\_typing.Tensors, mask: Optional[langml.tensor\_typing.Tensors] = None) → List[Union[langml.tensor\_typing.Tensors, None]]

**call**(self, inputs: langml.tensor\_typing.Tensors) → List[langml.tensor\_typing.Tensors]

**compute\_output\_shape**(self, input\_shape: langml.tensor\_typing.Tensors) → List[langml.tensor\_typing.Tensors]

**class** langml.prompt.models.ptuning.**PTuningPrompt**(plm\_backbone: str, plm\_config\_path: str, plm\_ckpt\_path: str, template: langml.prompt.base.Template, learning\_rate: float = 1e-05, freeze\_plm: bool = True, encoder: str = 'mlp')

Bases: langml.prompt.base.BasePromptModel

**build\_model**(self) → langml.tensor\_typing.Models

## Package Contents

### Classes

---

*PartialEmbedding*

---

*PTuningPrompt*

---

### Attributes

---

*custom\_objects*

---

**class** langml.prompt.models.**PartialEmbedding**(input\_dim: int, output\_dim: int, active\_start: int, active\_end: int, embeddings\_initializer: Optional[langml.tensor\_typing.Initializer] = 'uniform', embeddings\_regularizer: Optional[langml.tensor\_typing.Regularizer] = None, activity\_regularizer: Optional[langml.tensor\_typing.Regularizer] = None, embeddings\_constraint: Optional[langml.tensor\_typing.Constraint] = None, mask\_zero: bool = False, input\_length: Optional[int] = None, \*\*kwargs)

Bases: langml.L.Embedding

**static** get\_custom\_objects() → dict

**compute\_mask**(self, inputs: langml.tensor\_typing.Tensors, mask: Optional[langml.tensor\_typing.Tensors] = None) → List[Union[langml.tensor\_typing.Tensors, None]]

**call**(self, inputs: langml.tensor\_typing.Tensors) → List[langml.tensor\_typing.Tensors]

```
compute_output_shape(self, input_shape: langml.tensor_typing.Tensors) →  
List[langml.tensor_typing.Tensors]
```

```
class langml.prompt.models.PTuningPrompt(plm_backbone: str, plm_config_path: str, plm_ckpt_path:  
str, template: langml.prompt.base.Template, learning_rate:  
float = 1e-05, freeze_plm: bool = True, encoder: str = 'mlp')
```

Bases: [langml.prompt.base.BasePromptModel](#)

```
build_model(self) → langml.tensor_typing.Models
```

```
langml.prompt.models.custom_objects
```

## Submodules

```
langml.prompt.base
```

## Module Contents

### Classes

---

[Template](#)

---

[BasePromptModel](#)

---

[BasePromptTask](#)

---

[BaseDataGenerator](#)

---

```
class langml.prompt.base.Template(template: List[str], label_tokens_map: Dict[str, List[str]], tokenizer:  
langml.tokenizer.Tokenizer)
```

```
__len__(self) → int
```

```
encode_template(self, template: str) → List[int]
```

```
encode_label_tokens_map(self, label_tokens_map: Dict[str, List[str]]) → Dict[str, List[int]]
```

```
decode_label(self, idx: int, default='<UNK>') → str
```

```
class langml.prompt.base.BasePromptModel(plm_backbone: str, plm_config_path: str, plm_ckpt_path: str,  
template: Template, learning_rate: float = 1e-05, freeze_plm:  
bool = True)
```

```
abstract build_model(self) → langml.tensor_typing.Models
```

```
class langml.prompt.base.BasePromptTask(prompt_model: BasePromptModel, tokenizer:  
langml.tokenizer.Tokenizer)
```

```
abstract fit(self)
```

```
abstract predict(self)
```

```
class langml.prompt.base.BaseDataGenerator

    abstract make_iter(self, random: bool = False)

    abstract __len__(self)

    __call__(self, random: bool = False)
```

## Package Contents

### Classes

---

*Template*

---

*PTuningPrompt*

---

*PTuningForClassification*

---

```
class langml.prompt.Template(template: List[str], label_tokens_map: Dict[str, List[str]], tokenizer:
                               langml.tokenizer.Tokenizer)

    __len__(self) → int

    encode_template(self, template: str) → List[int]

    encode_label_tokens_map(self, label_tokens_map: Dict[str, List[str]]) → Dict[str, List[int]]

    decode_label(self, idx: int, default='<UNK>') → str

class langml.prompt.PTuningPrompt(plm_backbone: str, plm_config_path: str, plm_ckpt_path: str,
                                   template: langml.prompt.base.Template, learning_rate: float = 1e-05,
                                   freeze_plm: bool = True, encoder: str = 'mlp')

    Bases: langml.prompt.base.BasePromptModel

    build_model(self) → langml.tensor_typing.Models

class langml.prompt.PTuningForClassification(prompt_model: BasePromptModel, tokenizer:
                                              langml.tokenizer.Tokenizer)

    Bases: langml.prompt.base.BasePromptTask

    fit(self, data: List[str], labels: List[str], valid_data: Optional[List[str]] = None, valid_labels:
        Optional[List[str]] = None, model_path: Optional[str] = None, epoch: int = 20, batch_size: int = 16,
        early_stop: int = 10, do_shuffle: bool = True, f1_average: str = 'macro', verbose: int = 1)

        Fitting ptuning model for classification :param - data: List[str], texts of training data :param - labels:
        List[Union[str, List[str]]], training labels :param - valid_data: List[str], texts of valid data :param -
        valid_labels: List[Union[str, List[str]]], labels of valid data :param - model_path: Optional[str], path to
        save model, default None, do not to save model :param - epoch: int, epochs to train :param - batch_size:
        int, batch size, :param - early_stop: int, patience of early stop :param - do_shuffle: whether to shuffle data in
        training phase :param - f1_average: str, {'micro', 'macro', 'samples', 'weighted', 'binary'} or None :param
        - verbose: int, 0 = silent, 1 = progress bar, 2 = one line per epoch

    predict(self, text: str) → str
```

**load**(*self*, *model\_path*: str)

load model :param - model\_path: str, model path

**langml.third\_party**

## Submodules

**langml.third\_party.conlleval**

## Module Contents

### Classes

---

*EvalCounts*

---

### Functions

---

*parse\_args*(argv)

---

---

*parse\_tag*(t)

---

---

*evaluate*(iterable, options=None, delimiter=None)

---

---

*uniq*(iterable)

---

---

*calculate\_metrics*(correct, guessed, total)

---

---

*metrics*(counts)

---

---

*report*(counts, out=None)

---

---

*report\_notprint*(counts, out=None)

---

---

*end\_of\_chunk*(prev\_tag, tag, prev\_type, type\_)

---

---

*start\_of\_chunk*(prev\_tag, tag, prev\_type, type\_)

---

---

*return\_report*(input\_file)

---

---

*main*(argv)

---

## Attributes

---

*ANY\_SPACE*

---

*Metrics*

---

`langml.third_party.conlleva1.ANY_SPACE = <SPACE>`

**exception** `langml.third_party.conlleva1.FormatError`

Bases: Exception

Common base class for all non-exit exceptions.

`langml.third_party.conlleva1.Metrics`

**class** `langml.third_party.conlleva1.EvalCounts`

Bases: object

`langml.third_party.conlleva1.parse_args(argv)`

`langml.third_party.conlleva1.parse_tag(t)`

`langml.third_party.conlleva1.evaluate(iterable, options=None, delimiter=None)`

`langml.third_party.conlleva1.uniq(iterable)`

`langml.third_party.conlleva1.calculate_metrics(correct, guessed, total)`

`langml.third_party.conlleva1.metrics(counts)`

`langml.third_party.conlleva1.report(counts, out=None)`

`langml.third_party.conlleva1.report_notprint(counts, out=None)`

`langml.third_party.conlleva1.end_of_chunk(prev_tag, tag, prev_type, type_)`

`langml.third_party.conlleva1.start_of_chunk(prev_tag, tag, prev_type, type_)`

`langml.third_party.conlleva1.return_report(input_file)`

`langml.third_party.conlleva1.main(argv)`

`langml.third_party.crf`

## Module Contents

### Classes

---

*AbstractRNNCell*

Abstract object representing an RNN cell.

---

*CrfDecodeForwardRnnCell*

---

Computes the forward decoding in a linear-chain CRF.





## Functions

<code>viterbi_decode</code> (score: langml.tensor_typing.Tensors, langml.tensor_typing.Tensors) → ple[langml.tensor_typing.Tensors, langml.tensor_typing.Tensors]	trans: Tu-	<b>param score</b> A [seq_len, num_tags] matrix of unary potentials.
<code>_generate_zero_filled_state_for_cell</code> (cell, inputs, batch_size, dtype)		Generate a zero filled tensor with shape [batch_size, state_size].
<code>crf_filtered_inputs</code> (inputs: langml.tensor_typing.Tensors, langml.tensor_typing.Tensors) → tensorflow.Tensor	tag_bitmap:	Constrains the inputs to filter out certain tags at each time step.
<code>crf_sequence_score</code> (inputs: langml.tensor_typing.Tensors, langml.tensor_typing.Tensors, langml.tensor_typing.Tensors, langml.tensor_typing.Tensors) → tensorflow.Tensor	tag_indices: sequence_lengths: transition_params:	Computes the unnormalized score for a tag sequence.
<code>crf_multitag_sequence_score</code> (inputs: langml.tensor_typing.Tensors, langml.tensor_typing.Tensors, langml.tensor_typing.Tensors, langml.tensor_typing.Tensors) → tensorflow.Tensor	tag_bitmap: sequence_lengths: transition_params:	Computes the unnormalized score of all tag sequences matching
<code>crf_log_norm</code> (inputs: langml.tensor_typing.Tensors, sequence_lengths: langml.tensor_typing.Tensors, tran- sition_params: langml.tensor_typing.Tensors) → ten- sorflow.Tensor		Computes the normalization for a CRF.
<code>crf_log_likelihood</code> (inputs: langml.tensor_typing.Tensors, langml.tensor_typing.Tensors, langml.tensor_typing.Tensors, Optional[langml.tensor_typing.Tensors] = None) → tensorflow.Tensor	tag_indices: sequence_lengths: transition_params:	Computes the log-likelihood of tag sequences in a CRF.
<code>crf_unary_score</code> (tag_indices: langml.tensor_typing.Tensors, langml.tensor_typing.Tensors, langml.tensor_typing.Tensors) → tensorflow.Tensor	sequence_lengths: inputs:	Computes the unary scores of tag sequences.
<code>crf_binary_score</code> (tag_indices: langml.tensor_typing.Tensors, langml.tensor_typing.Tensors, langml.tensor_typing.Tensors) → tensorflow.Tensor	sequence_lengths: transition_params:	Computes the binary scores of tag sequences.
<code>crf_forward</code> (inputs: langml.tensor_typing.Tensors, state: langml.tensor_typing.Tensors, transition_params: langml.tensor_typing.Tensors, langml.tensor_typing.Tensors) → tensorflow.Tensor	sequence_lengths:	Computes the alpha values in a linear-chain CRF.
<code>crf_decode_forward</code> (inputs: langml.tensor_typing.Tensors, langml.tensor_typing.Tensors, langml.tensor_typing.Tensors, langml.tensor_typing.Tensors) → tensorflow.Tensor	state: transition_params: sequence_lengths:	Computes forward decoding in a linear-chain CRF.
<code>crf_decode_backward</code> (inputs: langml.tensor_typing.Tensors, langml.tensor_typing.Tensors) → tensorflow.Tensor	state:	Computes backward decoding in a linear-chain CRF.
<code>crf_decode</code> (potentials: langml.tensor_typing.Tensors, transition_params: langml.tensor_typing.Tensors, se- quence_length: langml.tensor_typing.Tensors) → ten- sorflow.Tensor		Decode the highest scoring sequence of tags.
<code>crf_constrained_decode</code> (potentials: langml.tensor_typing.Tensors, langml.tensor_typing.Tensors, langml.tensor_typing.Tensors)	tag_bitmap: transition_params:	Decode the highest scoring sequence of tags under con- straints.

```
langml.third_party.crf.viterbi_decode(score: langml.tensor_typing.Tensors, trans:
    langml.tensor_typing.Tensors) →
    Tuple[langml.tensor_typing.Tensors,
    langml.tensor_typing.Tensors]
```

#### Parameters

- **score** – A [seq\_len, num\_tags] matrix of unary potentials.
- **trans** – A [num\_tags, num\_tags] matrix of binary potentials.

#### Returns

A [seq\_len] list of integers containing the highest scoring tag indices.

viterbi\_score: A float containing the score for the Viterbi sequence.

#### Return type

viterbi

```
langml.third_party.crf._generate_zero_filled_state_for_cell(cell, inputs, batch_size, dtype)
    Generate a zero filled tensor with shape [batch_size, state_size].
```

```
langml.third_party.crf.crf_filtered_inputs(inputs: langml.tensor_typing.Tensors, tag_bitmap:
    langml.tensor_typing.Tensors) → tensorflow.Tensor
```

Constrains the inputs to filter out certain tags at each time step. tag\_bitmap limits the allowed tags at each input time step. This is useful when an observed output at a given time step needs to be constrained to a selected set of tags. Args: inputs: A [batch\_size, max\_seq\_len, num\_tags] tensor of unary potentials

to use as input to the CRF layer.

**tag\_bitmap: A [batch\_size, max\_seq\_len, num\_tags] boolean tensor**  
representing all active tags at each index for which to calculate the unnormalized score.

Returns: filtered\_inputs: A [batch\_size] vector of unnormalized sequence scores.

```
langml.third_party.crf.crf_sequence_score(inputs: langml.tensor_typing.Tensors, tag_indices:
    langml.tensor_typing.Tensors, sequence_lengths:
    langml.tensor_typing.Tensors, transition_params:
    langml.tensor_typing.Tensors) → tensorflow.Tensor
```

Computes the unnormalized score for a tag sequence. :param inputs: A [batch\_size, max\_seq\_len, num\_tags] tensor of unary potentials

to use as input to the CRF layer.

#### Parameters

- **tag\_indices** – A [batch\_size, max\_seq\_len] matrix of tag indices for which we compute the unnormalized score.
- **sequence\_lengths** – A [batch\_size] vector of true sequence lengths.
- **transition\_params** – A [num\_tags, num\_tags] transition matrix.

#### Returns

A [batch\_size] vector of unnormalized sequence scores.

#### Return type

sequence\_scores

```
langml.third_party.crf.crf_multitag_sequence_score(inputs: langml.tensor_typing.Tensors,  
                                                    tag_bitmap: langml.tensor_typing.Tensors,  
                                                    sequence_lengths: langml.tensor_typing.Tensors,  
                                                    transition_params:  
langml.tensor_typing.Tensors) →  
tensorflow.Tensor
```

Computes the unnormalized score of all tag sequences matching tag\_bitmap. tag\_bitmap enables more than one tag to be considered correct at each time step. This is useful when an observed output at a given time step is consistent with more than one tag, and thus the log likelihood of that observation must take into account all possible consistent tags. Using one-hot vectors in tag\_bitmap gives results identical to crf\_sequence\_score. :param inputs: A [batch\_size, max\_seq\_len, num\_tags] tensor of unary potentials

to use as input to the CRF layer.

#### Parameters

- **tag\_bitmap** – A [batch\_size, max\_seq\_len, num\_tags] boolean tensor representing all active tags at each index for which to calculate the unnormalized score.
- **sequence\_lengths** – A [batch\_size] vector of true sequence lengths.
- **transition\_params** – A [num\_tags, num\_tags] transition matrix.

#### Returns

A [batch\_size] vector of unnormalized sequence scores.

#### Return type

sequence\_scores

```
langml.third_party.crf.crf_log_norm(inputs: langml.tensor_typing.Tensors, sequence_lengths:  
langml.tensor_typing.Tensors, transition_params:  
langml.tensor_typing.Tensors) → tensorflow.Tensor
```

Computes the normalization for a CRF. :param inputs: A [batch\_size, max\_seq\_len, num\_tags] tensor of unary potentials

to use as input to the CRF layer.

#### Parameters

- **sequence\_lengths** – A [batch\_size] vector of true sequence lengths.
- **transition\_params** – A [num\_tags, num\_tags] transition matrix.

#### Returns

A [batch\_size] vector of normalizers for a CRF.

#### Return type

log\_norm

```
langml.third_party.crf.crf_log_likelihood(inputs: langml.tensor_typing.Tensors, tag_indices:  
langml.tensor_typing.Tensors, sequence_lengths:  
langml.tensor_typing.Tensors, transition_params:  
Optional[langml.tensor_typing.Tensors] = None) →  
tensorflow.Tensor
```

Computes the log-likelihood of tag sequences in a CRF. :param inputs: A [batch\_size, max\_seq\_len, num\_tags] tensor of unary potentials

to use as input to the CRF layer.

**Parameters**

- **tag\_indices** – A [batch\_size, max\_seq\_len] matrix of tag indices for which we compute the log-likelihood.
- **sequence\_lengths** – A [batch\_size] vector of true sequence lengths.
- **transition\_params** – A [num\_tags, num\_tags] transition matrix, if available.

**Returns**

A [batch\_size] *Tensor* containing the log-likelihood of each example, given the sequence of tag indices.

**transition\_params:** A [num\_tags, num\_tags] transition matrix. This is either provided by the caller or created in this function.

**Return type**

log\_likelihood

```
langml.third_party.crf.crf_unary_score(tag_indices: langml.tensor_typing.Tensors, sequence_lengths:
                                       langml.tensor_typing.Tensors, inputs:
                                       langml.tensor_typing.Tensors) → tensorflow.Tensor
```

Computes the unary scores of tag sequences. :param tag\_indices: A [batch\_size, max\_seq\_len] matrix of tag indices. :param sequence\_lengths: A [batch\_size] vector of true sequence lengths. :param inputs: A [batch\_size, max\_seq\_len, num\_tags] tensor of unary potentials.

**Returns**

A [batch\_size] vector of unary scores.

**Return type**

unary\_scores

```
langml.third_party.crf.crf_binary_score(tag_indices: langml.tensor_typing.Tensors, sequence_lengths:
                                         langml.tensor_typing.Tensors, transition_params:
                                         langml.tensor_typing.Tensors) → tensorflow.Tensor
```

Computes the binary scores of tag sequences. :param tag\_indices: A [batch\_size, max\_seq\_len] matrix of tag indices. :param sequence\_lengths: A [batch\_size] vector of true sequence lengths. :param transition\_params: A [num\_tags, num\_tags] matrix of binary potentials.

**Returns**

A [batch\_size] vector of binary scores.

**Return type**

binary\_scores

```
langml.third_party.crf.crf_forward(inputs: langml.tensor_typing.Tensors, state:
                                     langml.tensor_typing.Tensors, transition_params:
                                     langml.tensor_typing.Tensors, sequence_lengths:
                                     langml.tensor_typing.Tensors) → tensorflow.Tensor
```

Computes the alpha values in a linear-chain CRF. See <http://www.cs.columbia.edu/~mccollins/fb.pdf> for reference. :param inputs: A [batch\_size, num\_tags] matrix of unary potentials. :param state: A [batch\_size, num\_tags] matrix containing the previous alpha

values.

**Parameters**

- **transition\_params** – A [num\_tags, num\_tags] matrix of binary potentials. This matrix is expanded into a [1, num\_tags, num\_tags] in preparation for the broadcast summation occurring within the cell.

- **sequence\_lengths** – A [batch\_size] vector of true sequence lengths.

**Returns**

A [batch\_size, num\_tags] matrix containing the new alpha values.

**Return type**

new\_alphas

**class** langml.third\_party.crf.**AbstractRNNCell**

Bases: tensorflow.keras.layers.Layer

Abstract object representing an RNN cell. This is the base class for implementing RNN cells with custom behavior. Every *RNNCell* must have the properties below and implement *call* with the signature (*output*, *next\_state*) = *call*(*input*, *state*). Examples: ```python

```
class MinimalRNNCell(AbstractRNNCell): def __init__(self, units, **kwargs):
```

```
    self.units = units super(MinimalRNNCell, self).__init__(**kwargs)
```

```
@property def state_size(self):
```

```
    return self.units
```

```
def build(self, input_shape):
```

```
    self.kernel = self.add_weight(shape=(input_shape[-1], self.units),
                                  initializer='uniform', name='kernel')
```

```
    self.recurrent_kernel = self.add_weight(
        shape=(self.units, self.units), initializer='uniform', name='recurrent_kernel')
```

```
    self.built = True
```

```
def call(self, inputs, states):
```

```
    prev_output = states[0] h = K.dot(inputs, self.kernel) output = h + K.dot(prev_output,
    self.recurrent_kernel) return output, output
```

``` This definition of cell differs from the definition used in the literature. In the literature, ‘cell’ refers to an object with a single scalar output. This definition refers to a horizontal array of such units. An RNN cell, in the most abstract setting, is anything that has a state and performs some operation that takes a matrix of inputs. This operation results in an output matrix with *self.output\_size* columns. If *self.state\_size* is an integer, this operation also results in a new state matrix with *self.state\_size* columns. If *self.state\_size* is a (possibly nested tuple of) TensorShape object(s), then it should return a matching structure of Tensors having shape *[batch\_size].concatenate(s)* for each *s* in *self.batch\_size*.

**abstract call**(*self*, *inputs*, *states*)

The function that contains the logic for one RNN step calculation. Args: inputs: the input tensor, which is a slide from the overall RNN input by

the time dimension (usually the second dimension).

**states:** the state tensor from previous step, which has the same shape

as (*batch*, *state\_size*). In the case of timestep 0, it will be the initial state user specified, or zero filled tensor otherwise.

Returns: A tuple of two tensors:

1. output tensor for the current timestep, with size *output\_size*.

2. state tensor for next step, which has the shape of *state\_size*.

**property state\_size**(*self*)

size(s) of state(s) used by this cell. It can be represented by an Integer, a TensorShape or a tuple of Integers or TensorShapes.

**property output\_size**(*self*)

Integer or TensorShape: size of outputs produced by this cell.

**get\_initial\_state**(*self*, *inputs=None*, *batch\_size=None*, *dtype=None*)

**class** langml.third\_party.crf.CrfDecodeForwardRnnCell(*transition\_params*:  
langml.tensor\_typing.Tensors, *\*\*kwargs*)

Bases: [AbstractRnnCell](#)

Computes the forward decoding in a linear-chain CRF.

**property state\_size**(*self*)

size(s) of state(s) used by this cell. It can be represented by an Integer, a TensorShape or a tuple of Integers or TensorShapes.

**property output\_size**(*self*)

Integer or TensorShape: size of outputs produced by this cell.

**build**(*self*, *input\_shape*)

**compute\_mask**(*self*, *inputs*: langml.tensor\_typing.Tensors, *mask*: Optional[langml.tensor\_typing.Tensors] = None) → Union[List[Union[langml.tensor\_typing.Tensors, None]], langml.tensor\_typing.Tensors]

**call**(*self*, *inputs*: langml.tensor\_typing.Tensors, *state*: langml.tensor\_typing.Tensors, *mask*: Optional[langml.tensor\_typing.Tensors] = None, *\*\*kwargs*)

Build the CrfDecodeForwardRnnCell. :param inputs: A [batch\_size, num\_tags] matrix of unary potentials.  
:param state: A [batch\_size, num\_tags] matrix containing the previous step's  
score values.

#### Returns

A [batch\_size, num\_tags] matrix of backpointers. new\_state: A [batch\_size, num\_tags] matrix of new score values.

#### Return type

backpointers

**get\_config**(*self*) → dict

**classmethod from\_config**(*cls*, *config*: dict) → [CrfDecodeForwardRnnCell](#)

langml.third\_party.crf.crf\_decode\_forward(*inputs*: langml.tensor\_typing.Tensors, *state*:  
langml.tensor\_typing.Tensors, *transition\_params*:  
langml.tensor\_typing.Tensors, *sequence\_lengths*:  
langml.tensor\_typing.Tensors) → tensorflow.Tensor

Computes forward decoding in a linear-chain CRF. :param inputs: A [batch\_size, num\_tags] matrix of unary potentials. :param state: A [batch\_size, num\_tags] matrix containing the previous step's  
score values.

#### Parameters

- **transition\_params** – A [num\_tags, num\_tags] matrix of binary potentials.
- **sequence\_lengths** – A [batch\_size] vector of true sequence lengths.

**Returns**

A [batch\_size, num\_tags] matrix of backpointers. new\_state: A [batch\_size, num\_tags] matrix of new score values.

**Return type**

backpointers

`langml.third_party.crf.crf_decode_backward`(inputs: *langml.tensor\_typing.Tensors*, state: *langml.tensor\_typing.Tensors*) → tensorflow.Tensor

Computes backward decoding in a linear-chain CRF. :param inputs: A [batch\_size, num\_tags] matrix of backpointer of next step (in time order).

**Parameters**

**state** – A [batch\_size, 1] matrix of tag index of next step.

**Returns**

**A [batch\_size, num\_tags]**  
tensor containing the new tag indices.

**Return type**

new\_tags

`langml.third_party.crf.crf_decode`(potentials: *langml.tensor\_typing.Tensors*, transition\_params: *langml.tensor\_typing.Tensors*, sequence\_length: *langml.tensor\_typing.Tensors*) → tensorflow.Tensor

Decode the highest scoring sequence of tags. :param potentials: A [batch\_size, max\_seq\_len, num\_tags] tensor of

unary potentials.

**Parameters**

- **transition\_params** – A [num\_tags, num\_tags] matrix of binary potentials.
- **sequence\_length** – A [batch\_size] vector of true sequence lengths.

**Returns**

**A [batch\_size, max\_seq\_len] matrix, with dtype tf.int32.**  
Contains the highest scoring tag indices.

best\_score: A [batch\_size] vector, containing the score of *decode\_tags*.

**Return type**

decode\_tags

`langml.third_party.crf.crf_constrained_decode`(potentials: *langml.tensor\_typing.Tensors*, tag\_bitmap: *langml.tensor\_typing.Tensors*, transition\_params: *langml.tensor\_typing.Tensors*, sequence\_length: *langml.tensor\_typing.Tensors*) → tensorflow.Tensor

Decode the highest scoring sequence of tags under constraints. This is a function for tensor. :param potentials: A [batch\_size, max\_seq\_len, num\_tags] tensor of

unary potentials.



**Parameters**

- **tag\_bitmap** – A [batch\_size, max\_seq\_len, num\_tags] boolean tensor representing all active tags at each index for which to calculate the unnormalized score.
- **transition\_params** – A [num\_tags, num\_tags] matrix of binary potentials.
- **sequence\_length** – A [batch\_size] vector of true sequence lengths.

**Returns**

A [batch\_size, max\_seq\_len] matrix, with dtype *tf.int32*.

Contains the highest scoring tag indices.

best\_score: A [batch\_size] vector, containing the score of *decode\_tags*.

**Return type**

decode\_tags

`langml.transformer`

**Submodules**

`langml.transformer.encoder`

Yet another transformer implementation.

**Module Contents****Classes**


---

*TransformerEncoder*

---

*TransformerEncoderBlock*

---

```
class langml.transformer.encoder.TransformerEncoder(attention_heads: int, hidden_dim: int,
  attention_activation:
  langml.tensor_typing.Activation = None,
  feed_forward_activation:
  langml.tensor_typing.Activation = gelu,
  dropout_rate: float = 0.0, trainable: bool =
  True, name: str = 'Transformer-Encoder')

    __call__(self, inputs: langml.tensor_typing.Tensors) → langml.tensor_typing.Tensors

class langml.transformer.encoder.TransformerEncoderBlock(blocks: int, attention_heads: int,
   hidden_dim: int, attention_activation:
   langml.tensor_typing.Activation = None,
   feed_forward_activation:
   langml.tensor_typing.Activation = gelu,
   dropout_rate: float = 0.0, trainable: bool
   = False, name: str =
   'TransformerEncoderBlock',
   share_weights: bool = False)
```

```
__call__(self, inputs: langml.tensor_typing.Tensors) → langml.tensor_typing.Tensors
```

## langml.transformer.layers

Yet another transformer implementation.

## Module Contents

### Classes

| <i>FeedForward</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | Feed Forward Layer |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------|
| <pre>class langml.transformer.layers.FeedForward(units, activation: langml.tensor_typing.Activation = 'relu',                                              kernel_initializer: langml.tensor_typing.Initializer =                                              'glorot_normal', kernel_regularizer:                                              Optional[langml.tensor_typing.Regularizer] = None,                                              kernel_constraint:                                              Optional[langml.tensor_typing.Constraint] = None,                                              bias_initializer: langml.tensor_typing.Initializer = 'zeros',                                              bias_regularizer:                                              Optional[langml.tensor_typing.Regularizer] = None,                                              bias_constraint:                                              Optional[langml.tensor_typing.Constraint] = None,                                              use_bias: bool = True, dropout_rate: float = 0.0,                                              **kwargs)</pre> |                    |
| Bases: tensorflow.keras.layers.Layer                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                    |
| Feed Forward Layer <a href="https://arxiv.org/pdf/1706.03762.pdf">https://arxiv.org/pdf/1706.03762.pdf</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |                    |
| <b>get_config</b> (self) → dict                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |                    |
| <b>build</b> (self, input_shape: langml.tensor_typing.Tensors)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |                    |
| <b>call</b> (self, inputs: langml.tensor_typing.Tensors, mask: Optional[langml.tensor_typing.Tensors] = None,       training: Optional[Any] = None, **kwargs) → Union[List[langml.tensor_typing.Tensors],       langml.tensor_typing.Tensors]                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |                    |
| <b>compute_mask</b> (self, inputs: langml.tensor_typing.Tensors, mask:       Optional[Union[langml.tensor_typing.Tensors, List[langml.tensor_typing.Tensors]]] =       None) → Union[List[Union[langml.tensor_typing.Tensors, None]],       langml.tensor_typing.Tensors]                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                    |
| <b>static get_custom_objects</b> () → dict                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |                    |
| <b>compute_output_shape</b> (self, input_shape: langml.tensor_typing.Tensors) → langml.tensor_typing.Tensors                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |                    |

## Package Contents

### Classes

---

|                    |                    |
|--------------------|--------------------|
| <i>FeedForward</i> | Feed Forward Layer |
|--------------------|--------------------|

---

### Attributes

---

|                       |
|-----------------------|
| <i>TF_KERAS</i>       |
| <i>custom_objects</i> |

---

langml.transformer.TF\_KERAS

```
class langml.transformer.FeedForward(units, activation: langml.tensor_typing.Activation = 'relu',
                                     kernel_initializer: langml.tensor_typing.Initializer =
                                     'glorot_normal', kernel_regularizer:
                                     Optional[langml.tensor_typing.Regularizer] = None,
                                     kernel_constraint: Optional[langml.tensor_typing.Constraint] =
                                     None, bias_initializer: langml.tensor_typing.Initializer = 'zeros',
                                     bias_regularizer: Optional[langml.tensor_typing.Regularizer] =
                                     None, bias_constraint: Optional[langml.tensor_typing.Constraint]
                                     = None, use_bias: bool = True, dropout_rate: float = 0.0,
                                     **kwargs)
```

Bases: tensorflow.keras.layers.Layer

Feed Forward Layer <https://arxiv.org/pdf/1706.03762.pdf>

**get\_config**(self) → dict

**build**(self, input\_shape: langml.tensor\_typing.Tensors)

**call**(self, inputs: langml.tensor\_typing.Tensors, mask: Optional[langml.tensor\_typing.Tensors] = None,
 training: Optional[Any] = None, \*\*kwargs) → Union[List[langml.tensor\_typing.Tensors],
 langml.tensor\_typing.Tensors]

**compute\_mask**(self, inputs: langml.tensor\_typing.Tensors, mask:
 Optional[Union[langml.tensor\_typing.Tensors, List[langml.tensor\_typing.Tensors]]] =
 None) → Union[List[Union[langml.tensor\_typing.Tensors, None]],
 langml.tensor\_typing.Tensors]

**static get\_custom\_objects**() → dict

**compute\_output\_shape**(self, input\_shape: langml.tensor\_typing.Tensors) → langml.tensor\_typing.Tensors

langml.transformer.custom\_objects

## 6.1.2 Submodules

### langml.activations

Activations

#### Module Contents

#### Functions

---

|                                                |   |                                     |
|------------------------------------------------|---|-------------------------------------|
| <i>gelu</i> (x: langml.tensor_typing.Tensors)  | → | Gaussian Error Linear Units (GELUs) |
| langml.tensor_typing.Tensors                   |   |                                     |
| <i>relu2</i> (x: langml.tensor_typing.Tensors) | → |                                     |
| langml.tensor_typing.Tensors                   |   |                                     |

---

#### Attributes

---

|                       |
|-----------------------|
| <i>custom_objects</i> |
|-----------------------|

---

langml.activations.**gelu**(x: *langml.tensor\_typing.Tensors*) → langml.tensor\_typing.Tensors

Gaussian Error Linear Units (GELUs) <https://arxiv.org/abs/1606.08415>

$\text{\$GELU}(x) = 0.5x(1 + \tanh[\text{sqrt}(2 / \text{Pi}) (x + 0.044715x^3)])\text{\$}$

langml.activations.**relu2**(x: *langml.tensor\_typing.Tensors*) → langml.tensor\_typing.Tensors

langml.activations.**custom\_objects**

### langml.cli

#### Module Contents

#### Functions

---

|                |               |
|----------------|---------------|
| <i>cli</i> ()  | LangML client |
| <i>main</i> () |               |

---

langml.cli.**cli**()

LangML client

langml.cli.**main**()

---

`langml.log`

## Module Contents

### Functions

---

`print_log(level: int, msg: str, *args)`

---

### Attributes

---

`debug`

---

---

`info`

---

---

`warn`

---

---

`error`

---

`langml.log.print_log(level: int, msg: str, *args)``langml.log.debug``langml.log.info``langml.log.warn``langml.log.error``langml.model`

## Module Contents

### Functions

---

`get_random_string(length)`

---

---

`export_model_v1(model, export_model_dir)`

---

**param export\_model\_dir**  
type string, save dir for exported  
model url

---

`save_frozen(model: langml.tensor_typing.Models,  
fpath: str)`

---

---

`load_frozen(model_dir: str, session: Any = None) →  
Any`

---

## Attributes

---

### `SAVED_MODEL_TAG`

---

```
langml.model.SAVED_MODEL_TAG = serve
langml.model.get_random_string(length)
langml.model.export_model_v1(model, export_model_dir)
```

#### Parameters

- **export\_model\_dir** – type string, save dir for exported model url
- **model\_version** – type int best

:return:no return

```
langml.model.save_frozen(model: langml.tensor_typing.Models, fpath: str)
```

```
langml.model.load_frozen(model_dir: str, session: Any = None) → Any
```

## `langml.tensor_typing`

### Module Contents

```
langml.tensor_typing.Number
langml.tensor_typing.Initializer
langml.tensor_typing.Regularizer
langml.tensor_typing.Constraint
langml.tensor_typing.Activation
langml.tensor_typing.Optimizer
langml.tensor_typing.Tensors
langml.tensor_typing.Models
```

## `langml.tokenizer`

LangML Tokenizer

- WPTokenizer: WordPiece Tokenizer
- SPTokenizer: SentencePiece Tokenizer

Wrap for:

- tokenizers.BertWordPieceTokenizer
- sentencepiece.SentencePieceProcessor

We don't provide all functions of raw tokenizer, please use raw tokenizer for full usage.

## Module Contents

### Classes

|                      |                               |
|----------------------|-------------------------------|
| <i>Encoding</i>      | Product of tokenizer encoding |
| <i>SpecialTokens</i> |                               |
| <i>Tokenizer</i>     | Base Tokenizer                |
| <i>SPTokenizer</i>   | SentencePiece Tokenizer       |
| <i>WPTokenizer</i>   | WordPieceTokenizer            |

```
class langml.tokenizer.Encoding(ids: Union[numpy.ndarray, List[int]], segment_ids:
                                Union[numpy.ndarray, List[int]], tokens: List[str])
```

Product of tokenizer encoding

**ids**

**segment\_ids**

**tokens**

```
class langml.tokenizer.SpecialTokens
```

**PAD** = [PAD]

**UNK** = [UNK]

**MASK** = [MASK]

**CLS** = [CLS]

**SEP** = [SEP]

**\_\_contains\_\_**(self, token: str) → bool

Check if the input token exists in special tokens. :param - token: str

**Returns**

bool

**tokens**(self) → List[str]

```
class langml.tokenizer.Tokenizer(vocab_path: str, lowercase: bool = False)
```

Base Tokenizer

**enable\_truncation**(self, max\_length: int, strategy: str = 'post')

**Parameters**

- **max\_length** (-) – int,
- **strategy** (-) – str, optional, truncation strategy, options: *post* or *pre*, default *post*

**tokens\_mapping**(self, sequence: str, tokens: List[str]) → List[Tuple[int, int]]

Get tokens to their corresponding sequence position mapping. Tokens may contain special marks, e.g., ##, , and [UNK]. Use this function can obtain the corresponding raw token in the sequence.

**Parameters**

- **sequence** (-) – str, the input sequence

- **tokens** (-) – List[str], tokens of the input sequence

**Returns**

List[Tuple[int, int]]

Examples: >>> sequence = 'I like watermelons' >>> tokens = ['[CLS]', 'i', 'like', 'water', 'mel', 'ons', '[SEP]'] >>> mapping = tokenizer.tokens\_mapping(tokens) >>> start\_index, end\_index = 3, 5 >>> print("current token", tokens[start\_index: end\_index + 1]) ['water', 'mel', 'ons'] >>> print("raw token", sequence[mapping[start\_index][0]: mapping[end\_index][1]]) watermelons

**Reference:**

<https://github.com/bojone/bert4keras>

**encode**(self, sequence: str, pair: Optional[str] = None, return\_array: bool = False) → *Encoding*

**Parameters**

- **sequence** (-) – str, input sequence
- **pair** (-) – str, optional, pair sequence, default *None*
- **return\_array** (-) – bool, optional, whether to return numpy array, default *True*

**Returns**

Encoding object

**encode\_batch**(self, inputs: Union[List[str], List[Tuple[str, str]], List[List[str]]], padding: bool = True, padding\_strategy: str = 'post', return\_array: bool = False) → *Encoding*

**Parameters**

- **inputs** (-) – Union[List[str], List[Tuple[str, str]], List[List[str]]], list of texts or list of text pairs.
- **padding** (-) – bool, optional, whether to padding sequences, default *True*
- **padding\_strategy** (-) – str, optional, options: *post* or *pre*, default *post*
- **return\_array** (-) – bool, optional, whether to return numpy array, default *True*

**Returns**

Encoding object

**stem**(self, token)

**sequence\_lower**(self, sequence: str) → str

Do lower to sequence, except for special tokens. :param - sequence: str

**Returns**

str

**sequence\_truncating**(self, max\_token\_length: int, tokens: List[str], pair\_tokens: Optional[List[str]] = None) → Tuple[List[str], Optional[List[str]]]

Truncating sequence :param - max\_token\_length: int, maximum token length :param - tokens: List[str], input tokens :param - pair\_tokens: Optional[List[str]], optional, input pair tokens, default *None*

**Returns**

Tuple[List[str], Optional[List[str]]]

**raw\_tokenizer**(self) → object

Return raw tokenizer, i.e. object of *tokenizers.BertWordPieceTokenizer* or *sentence-piece.SentencePieceProcessor*



```

abstract tokenize(self, sequence: str) → List[str]

abstract decode(self, ids: List[int], skip_special_tokens: bool = True) → List[str]

abstract get_vocab_size(self) → int

abstract id_to_token(self, idx: int) → str

abstract token_to_id(self, token: str) → int

abstract get_vocab(self) → Dict

class langml.tokenizer.SPTokenizer(vocab_path: str, lowercase: bool = False)
    Bases: Tokenizer
    SentencePiece Tokenizer Wrap for sentencepiece.

    get_vocab_size(self) → int
        Return vocab size

    token_to_id(self, token: str) → int
        Convert the input token to corresponding index :param - token: str

        Returns
        int

    id_to_token(self, idx: int) → str
        Convert index to corresponding token :param - idx: int

        Returns
        str

    tokenize(self, sequence: str) → List[str]
        Tokenize sequence to token peices. :param - sequence: str

        Returns
        List[str]

    decode(self, ids: List[int], skip_special_tokens: bool = True) → List[str]
        Decode indexs to tokens :param - ids: List[int] :param - skip_special_tokens: bool, optioanl, whether to
        skip special tokens, default True

        Returns
        List[str]

    get_vocab(self) → Dict
        Return vocabulary

class langml.tokenizer.WPTokenizer(vocab_path: str, lowercase: bool = False)
    Bases: Tokenizer
    WordPieceTokenizer Wrap for BertWordPieceTokenizer.

    get_vocab_size(self) → int
        Return vocab size

    token_to_id(self, token: str) → int
        Convert the input token to corresponding index :param - token: str

        Returns
        int

```

**id\_to\_token**(*self*, *idx*: int) → str

Convert index to corresponding token :param - idx: int

**Returns**

str

**tokenize**(*self*, *sequence*: str) → List[str]

Tokenize sequence to token peices. :param - sequence: str

**Returns**

List[str]

**decode**(*self*, *ids*: List[int], *skip\_special\_tokens*: bool = True) → List[str]

Decode indexs to tokens :param - ids: List[int] :param - skip\_special\_tokens: bool, optionl, whether to skip special tokens, default True

**Returns**

List[str]

**get\_vocab**(*self*) → Dict

Return vocabulary

**add\_special\_tokens**(*self*, *tokens*: List[str])

Specify special tokens, the tokenizer will reserve special tokens as a whole (i.e. don't split them) in tokenizing. Currently, only the WPTokenizer supports specifying special tokens. :param - tokens: List[str], special tokens

## langml.utils

### Module Contents

#### Functions

---

*deprecated\_warning*(*msg*='this function is deprecated! it might be removed in a future version.')

---

*bio\_decode*(*tags*: List[str]) → List[Tuple[int, int, str]] Decode BIO tags

---

*load\_variables*(*checkpoint\_path*: str) → Callable load variables from chechkpoint

---

*auto\_tokenizer*(*vocab\_path*: str, *lowercase*: bool = False) → langml.tokenizer.Tokenizer

---

langml.utils.**deprecated\_warning**(*msg*='this function is deprecated! it might be removed in a future version.')

langml.utils.**bio\_decode**(*tags*: List[str]) → List[Tuple[int, int, str]]

Decode BIO tags

Examples: >>> bio\_decode(['B-PER', 'I-PER', 'O', 'B-ORG', 'I-ORG', 'I-ORG']) >>> [(0, 1, 'PER'), (3, 5, 'ORG')]

langml.utils.**load\_variables**(*checkpoint\_path*: str) → Callable

load variables from chechkpoint

langml.utils.**auto\_tokenizer**(*vocab\_path*: str, *lowercase*: bool = False) → *langml.tokenizer.Tokenizer*

### 6.1.3 Package Contents

`langml.__version__ = 0.4.2`

`langml.TF_VERSION`

`langml.TF_KERAS`



## PYTHON MODULE INDEX

|  
langml, 17  
langml.activations, 72  
langml.baselines, 17  
langml.baselines.clf, 17  
langml.baselines.clf.bert, 17  
langml.baselines.clf.bilstm, 18  
langml.baselines.clf.cli, 18  
langml.baselines.clf.dataloader, 20  
langml.baselines.clf.textcnn, 20  
langml.baselines.cli, 32  
langml.baselines.contrastive, 21  
langml.baselines.contrastive.cli, 24  
langml.baselines.contrastive.simcse, 21  
langml.baselines.contrastive.simcse.dataloder, 21  
langml.baselines.contrastive.simcse.model, 22  
langml.baselines.contrastive.utils, 25  
langml.baselines.matching, 25  
langml.baselines.matching.cli, 28  
langml.baselines.matching.sbert, 25  
langml.baselines.matching.sbert.dataloder, 25  
langml.baselines.matching.sbert.model, 26  
langml.baselines.ner, 28  
langml.baselines.ner.bert\_crf, 28  
langml.baselines.ner.cli, 29  
langml.baselines.ner.dataloader, 30  
langml.baselines.ner.lstm\_crf, 30  
langml.cli, 72  
langml.common, 33  
langml.common.evaluator, 33  
langml.common.evaluator.spearman, 33  
langml.layers, 33  
langml.layers.attention, 33  
langml.layers.crf, 37  
langml.layers.layer\_norm, 38  
langml.layers.layers, 39  
langml.log, 73  
langml.model, 73  
langml.plm, 46  
langml.plm.albert, 46  
langml.plm.bert, 47  
langml.plm.layers, 49  
langml.prompt, 53  
langml.prompt.base, 57  
langml.prompt.clf, 53  
langml.prompt.clf.ptuning, 53  
langml.prompt.clf.utils, 54  
langml.prompt.models, 55  
langml.prompt.models.ptuning, 55  
langml.tensor\_typing, 74  
langml.third\_party, 59  
langml.third\_party.conlleval, 59  
langml.third\_party.crf, 60  
langml.tokenizer, 74  
langml.transformer, 69  
langml.transformer.encoder, 69  
langml.transformer.layers, 70  
langml.utils, 78



## Symbols

|                             |                                                                                        |                                                     |                                                                              |
|-----------------------------|----------------------------------------------------------------------------------------|-----------------------------------------------------|------------------------------------------------------------------------------|
| <code>__call__()</code>     | ( <code>langml.baselines.BaseDataLoader</code> method), 32                             | <code>__len__()</code>                              | ( <code>langml.prompt.base.Template</code> method), 57                       |
| <code>__call__()</code>     | ( <code>langml.baselines.clf.Infer</code> method), 21                                  | <code>__len__()</code>                              | ( <code>langml.prompt.clf.ptuning.DataGenerator</code> method), 53           |
| <code>__call__()</code>     | ( <code>langml.baselines.clf.dataloader.TFDataLoader</code> method), 20                | <code>version__</code>                              | (in module <code>langml</code> ), 79                                         |
| <code>__call__()</code>     | ( <code>langml.baselines.contrastive.simcse.TFDataLoader</code> method), 24            | <code>_attention_penalty()</code>                   | ( <code>langml.layers.SelfAdditiveAttention</code> method), 44               |
| <code>__call__()</code>     | ( <code>langml.baselines.contrastive.simcse.dataloader.TFDataLoader</code> method), 22 | <code>attention_penalty()</code>                    | ( <code>langml.layers.SelfAttention</code> method), 44                       |
| <code>__call__()</code>     | ( <code>langml.baselines.matching.sbert.TFDataLoader</code> method), 27                | <code>attention_penalty()</code>                    | ( <code>langml.layers.attention.SelfAdditiveAttention</code> method), 35     |
| <code>__call__()</code>     | ( <code>langml.baselines.matching.sbert.dataloader.TFDataLoader</code> method), 26     | <code>_attention_penalty()</code>                   | ( <code>langml.layers.attention.SelfAttention</code> method), 34             |
| <code>__call__()</code>     | ( <code>langml.baselines.ner.Infer</code> method), 31                                  | <code>_generate_zero_filled_state_for_cell()</code> | (in module <code>langml.third_party.crf</code> ), 63                         |
| <code>__call__()</code>     | ( <code>langml.baselines.ner.dataloader.TFDataLoader</code> method), 30                | <code>_reshape_attention_from_batches()</code>      | ( <code>langml.layers.MultiHeadAttention</code> static method), 45           |
| <code>__call__()</code>     | ( <code>langml.plm.bert.BERT</code> method), 48                                        | <code>_reshape_attention_from_batches()</code>      | ( <code>langml.layers.attention.MultiHeadAttention</code> static method), 36 |
| <code>__call__()</code>     | ( <code>langml.prompt.base.BaseDataGenerator</code> method), 58                        | <code>_reshape_from_batches()</code>                | ( <code>langml.layers.MultiHeadAttention</code> static method), 45           |
| <code>__call__()</code>     | ( <code>langml.transformer.encoder.TransformerEncoder</code> method), 69               | <code>_reshape_from_batches()</code>                | ( <code>langml.layers.attention.MultiHeadAttention</code> static method), 36 |
| <code>__call__()</code>     | ( <code>langml.transformer.encoder.TransformerEncoderBlock</code> method), 70          | <code>_reshape_mask()</code>                        | ( <code>langml.layers.MultiHeadAttention</code> static method), 45           |
| <code>__contains__()</code> | ( <code>langml.tokenizer.SpecialTokens</code> method), 75                              | <code>_reshape_mask()</code>                        | ( <code>langml.layers.attention.MultiHeadAttention</code> static method), 36 |
| <code>__len__()</code>      | ( <code>langml.baselines.BaseDataLoader</code> method), 32                             | <code>_reshape_to_batches()</code>                  | ( <code>langml.layers.MultiHeadAttention</code> static method), 45           |
| <code>__len__()</code>      | ( <code>langml.baselines.clf.dataloader.DataLoader</code> method), 20                  | <code>_reshape_to_batches()</code>                  | ( <code>langml.layers.attention.MultiHeadAttention</code> static method), 36 |
| <code>__len__()</code>      | ( <code>langml.baselines.contrastive.simcse.DataLoader</code> method), 23              | <code>_wrap()</code>                                | ( <code>langml.baselines.Parameters</code> method), 32                       |
| <code>__len__()</code>      | ( <code>langml.baselines.contrastive.simcse.dataloader.DataLoader</code> method), 22   |                                                     |                                                                              |
| <code>__len__()</code>      | ( <code>langml.baselines.matching.sbert.DataLoader</code> method), 27                  |                                                     |                                                                              |
| <code>__len__()</code>      | ( <code>langml.baselines.matching.sbert.dataloader.DataLoader</code> method), 26       |                                                     |                                                                              |
| <code>__len__()</code>      | ( <code>langml.baselines.ner.dataloader.DataLoader</code> method), 30                  |                                                     |                                                                              |
| <code>__len__()</code>      | ( <code>langml.prompt.Template</code> method), 58                                      |                                                     |                                                                              |
| <code>__len__()</code>      | ( <code>langml.prompt.base.BaseDataGenerator</code>                                    |                                                     |                                                                              |

## A

`AbsolutePositionEmbedding` (class in `langml.layers`), 42

`AbsolutePositionEmbedding` (class in `langml.layers.layers`), 39

`AbstractRNNCell` (class in `langml.third_party.crf`), 66

`accuracy` (`langml.layers.CRF` property), 41

`accuracy` (`langml.layers.crf.CRF` property), 38

`Activation` (in module `langml.tensor_typing`), 74

`add()` (`langml.baselines.Parameters` method), 32

`add_special_tokens()` (`langml.tokenizer.WPTokenizer` method), 78

`aeda_augment()` (in module `langml.baselines.contrastive.utils`), 25

`ANY_SPACE` (in module `langml.third_party.conlleval`), 60

`apply_rotary_position_embeddings()` (`langml.layers.attention.GatedAttentionUnit` method), 37

`apply_rotary_position_embeddings()` (`langml.layers.GatedAttentionUnit` method), 46

`attn()` (`langml.layers.attention.GatedAttentionUnit` method), 37

`attn()` (`langml.layers.GatedAttentionUnit` method), 46

`auto_tokenizer()` (in module `langml.utils`), 78

## B

`BaseDataGenerator` (class in `langml.prompt.base`), 57

`BaseDataLoader` (class in `langml.baselines`), 32

`baseline()` (in module `langml.baselines.cli`), 32

`BaselineModel` (class in `langml.baselines`), 32

`BasePromptModel` (class in `langml.prompt.base`), 57

`BasePromptTask` (class in `langml.prompt.base`), 57

`BERT` (class in `langml.plm.bert`), 48

`bert()` (in module `langml.baselines.clf.cli`), 19

`bert_crf()` (in module `langml.baselines.ner.cli`), 29

`BertClassifier` (class in `langml.baselines.clf.bert`), 17

`BertCRF` (class in `langml.baselines.ner.bert_crf`), 28

`bilstm()` (in module `langml.baselines.clf.cli`), 20

`BiLSTMClassifier` (class in `langml.baselines.clf.bilstm`), 18

`bio_decode()` (in module `langml.baselines.ner`), 31

`bio_decode()` (in module `langml.utils`), 78

`build()` (`langml.layers.AbsolutePositionEmbedding` method), 42

`build()` (`langml.layers.attention.GatedAttentionUnit` method), 37

`build()` (`langml.layers.attention.MultiHeadAttention` method), 36

`build()` (`langml.layers.attention.SelfAdditiveAttention` method), 35

`build()` (`langml.layers.attention.SelfAttention` method), 34

`build()` (`langml.layers.ConditionalLayerNormalization` method), 43

`build()` (`langml.layers.CRF` method), 41

`build()` (`langml.layers.crf.CRF` method), 38

`build()` (`langml.layers.GatedAttentionUnit` method), 46

`build()` (`langml.layers.layer_norm.LayerNorm` method), 38

`build()` (`langml.layers.LayerNorm` method), 42

`build()` (`langml.layers.layers.AbsolutePositionEmbedding` method), 39

`build()` (`langml.layers.layers.ConditionalLayerNormalization` method), 40

`build()` (`langml.layers.layers.ScaleOffset` method), 40

`build()` (`langml.layers.MultiHeadAttention` method), 45

`build()` (`langml.layers.ScaleOffset` method), 43

`build()` (`langml.layers.SelfAdditiveAttention` method), 44

`build()` (`langml.layers.SelfAttention` method), 43

`build()` (`langml.plm.bert.BERT` method), 48

`build()` (`langml.plm.EmbeddingMatching` method), 51

`build()` (`langml.plm.layers.EmbeddingMatching` method), 49

`build()` (`langml.third_party.crf.CrfDecodeForwardRnnCell` method), 67

`build()` (`langml.transformer.FeedForward` method), 71

`build()` (`langml.transformer.layers.FeedForward` method), 70

`build_model()` (`langml.baselines.BaselineModel` method), 32

`build_model()` (`langml.baselines.clf.bert.BertClassifier` method), 17

`build_model()` (`langml.baselines.clf.bilstm.BiLSTMClassifier` method), 18

`build_model()` (`langml.baselines.clf.textcnn.TextCNNClassifier` method), 20

`build_model()` (`langml.baselines.contrastive.simcse.model.SimCSE` method), 23

`build_model()` (`langml.baselines.contrastive.simcse.SimCSE` method), 24

`build_model()` (`langml.baselines.matching.sbert.model.SentenceBert` method), 26

`build_model()` (`langml.baselines.matching.sbert.SentenceBert` method), 27

`build_model()` (`langml.baselines.ner.bert_crf.BertCRF` method), 28

`build_model()` (`langml.baselines.ner.lstm_crf.LSTMCRF` method), 30

`build_model()` (`langml.prompt.base.BasePromptModel` method), 57

`build_model()` (`langml.prompt.models.ptuning.PTuningPrompt` method), 56

`build_model()` (`langml.prompt.models.PTuningPrompt` method), 57

`build_model()` (`langml.prompt.PTuningPrompt` method), 57



method), 58

## C

calculate\_metrics() (in module langml.third\_party.conlleval), 60

call() (langml.layers.AbsolutePositionEmbedding method), 42

call() (langml.layers.attention.GatedAttentionUnit method), 37

call() (langml.layers.attention.MultiHeadAttention method), 36

call() (langml.layers.attention.ScaledDotProductAttention method), 35

call() (langml.layers.attention.SelfAdditiveAttention method), 35

call() (langml.layers.attention.SelfAttention method), 34

call() (langml.layers.ConditionalLayerNormalization method), 43

call() (langml.layers.CRF method), 41

call() (langml.layers.crf.CRF method), 38

call() (langml.layers.GatedAttentionUnit method), 46

call() (langml.layers.layer\_norm.LayerNorm method), 38

call() (langml.layers.LayerNorm method), 42

call() (langml.layers.layers.AbsolutePositionEmbedding method), 39

call() (langml.layers.layers.ConditionalLayerNormalization method), 40

call() (langml.layers.layers.ScaleOffset method), 40

call() (langml.layers.layers.SineCosinePositionEmbedding method), 40

call() (langml.layers.MultiHeadAttention method), 45

call() (langml.layers.ScaledDotProductAttention method), 45

call() (langml.layers.ScaleOffset method), 43

call() (langml.layers.SelfAdditiveAttention method), 44

call() (langml.layers.SelfAttention method), 43

call() (langml.layers.SineCosinePositionEmbedding method), 42

call() (langml.plm.EmbeddingMatching method), 51

call() (langml.plm.layers.EmbeddingMatching method), 49

call() (langml.plm.layers.Masked method), 50

call() (langml.plm.layers.TokenEmbedding method), 49

call() (langml.plm.Masked method), 51

call() (langml.plm.TokenEmbedding method), 51

call() (langml.prompt.models.PartialEmbedding method), 56

call() (langml.prompt.models.ptuning.PartialEmbedding method), 56

call() (langml.third\_party.crf.AbstractRNCell method), 66

call() (langml.third\_party.crf.CrfDecodeForwardRnnCell method), 67

call() (langml.transformer.FeedForward method), 71

call() (langml.transformer.layers.FeedForward method), 70

clf() (in module langml.baselines.clf.cli), 19

cli() (in module langml.cli), 72

CLS (langml.tokenizer.SpecialTokens attribute), 75

CN\_PUNCTUATIONS (in module langml.baselines.contrastive.utils), 25

compute\_corrcoef() (langml.common.evaluator.spearman.SpearmanEvaluator method), 33

compute\_corrcoef() (langml.common.evaluator.SpearmanEvaluator method), 33

compute\_detail\_metrics() (in module langml.baselines.clf), 21

compute\_detail\_metrics() (in module langml.baselines.ner), 32

compute\_mask() (langml.layers.AbsolutePositionEmbedding method), 42

compute\_mask() (langml.layers.attention.GatedAttentionUnit method), 37

compute\_mask() (langml.layers.attention.MultiHeadAttention method), 36

compute\_mask() (langml.layers.attention.ScaledDotProductAttention method), 35

compute\_mask() (langml.layers.attention.SelfAdditiveAttention method), 35

compute\_mask() (langml.layers.attention.SelfAttention method), 34

compute\_mask() (langml.layers.ConditionalLayerNormalization method), 43

compute\_mask() (langml.layers.CRF method), 41

compute\_mask() (langml.layers.crf.CRF method), 38

compute\_mask() (langml.layers.GatedAttentionUnit method), 46

compute\_mask() (langml.layers.layer\_norm.LayerNorm method), 38

compute\_mask() (langml.layers.LayerNorm method), 42

compute\_mask() (langml.layers.layers.AbsolutePositionEmbedding method), 39

compute\_mask() (langml.layers.layers.ConditionalLayerNormalization method), 40

compute\_mask() (langml.layers.layers.ScaleOffset method), 40

compute\_mask() (langml.layers.layers.SineCosinePositionEmbedding method), 39

compute\_mask() (langml.layers.MultiHeadAttention method), 45

compute\_mask() (langml.layers.ScaledDotProductAttention method), 45

compute\_mask() (langml.layers.ScaleOffset method), 43

compute\_mask() (langml.layers.SelfAdditiveAttention method), 44

*method*), 44

`compute_mask()` (*langml.layers.SelfAttention method*), 44

`compute_mask()` (*langml.layers.SineCosinePositionEmbedding method*), 42

`compute_mask()` (*langml.plm.EmbeddingMatching method*), 51

`compute_mask()` (*langml.plm.layers.EmbeddingMatching method*), 49

`compute_mask()` (*langml.plm.layers.Masked method*), 50

`compute_mask()` (*langml.plm.layers.TokenEmbedding method*), 49

`compute_mask()` (*langml.plm.Masked method*), 51

`compute_mask()` (*langml.plm.TokenEmbedding method*), 51

`compute_mask()` (*langml.prompt.models.PartialEmbedding method*), 56

`compute_mask()` (*langml.prompt.models.ptuning.PartialEmbedding method*), 55

`compute_mask()` (*langml.third\_party.crf.CrfDecodeForwardRnnCell method*), 67

`compute_mask()` (*langml.transformer.FeedForward method*), 71

`compute_mask()` (*langml.transformer.layers.FeedForward method*), 70

`compute_output_shape()` (*langml.layers.AbsolutePositionEmbedding method*), 42

`compute_output_shape()` (*langml.layers.attention.GatedAttentionUnit method*), 37

`compute_output_shape()` (*langml.layers.attention.MultiHeadAttention method*), 36

`compute_output_shape()` (*langml.layers.attention.ScaledDotProductAttention method*), 36

`compute_output_shape()` (*langml.layers.attention.SelfAdditiveAttention method*), 35

`compute_output_shape()` (*langml.layers.attention.SelfAttention method*), 34

`compute_output_shape()` (*langml.layers.ConditionalLayerNormalization method*), 43

`compute_output_shape()` (*langml.layers.CRF method*), 41

`compute_output_shape()` (*langml.layers.crf.CRF method*), 38

`compute_output_shape()` (*langml.layers.GatedAttentionUnit method*), 46

`compute_output_shape()` (*langml.layers.layer\_norm.LayerNorm method*), 39

`compute_output_shape()` (*langml.layers.LayerNorm method*), 42

`compute_output_shape()` (*langml.layers.layers.AbsolutePositionEmbedding method*), 39

`compute_output_shape()` (*langml.layers.layers.ConditionalLayerNormalization method*), 40

`compute_output_shape()` (*langml.layers.layers.ScaleOffset method*), 40

`compute_output_shape()` (*langml.layers.layers.SineCosinePositionEmbedding method*), 39

`compute_output_shape()` (*langml.layers.MultiHeadAttention method*), 45

`compute_output_shape()` (*langml.layers.ScaledDotProductAttention method*), 45

`compute_output_shape()` (*langml.layers.ScaleOffset method*), 43

`compute_output_shape()` (*langml.layers.SelfAdditiveAttention method*), 44

`compute_output_shape()` (*langml.layers.SelfAttention method*), 44

`compute_output_shape()` (*langml.layers.SineCosinePositionEmbedding method*), 42

`compute_output_shape()` (*langml.plm.EmbeddingMatching method*), 51

`compute_output_shape()` (*langml.plm.layers.EmbeddingMatching method*), 49

`compute_output_shape()` (*langml.plm.layers.Masked method*), 50

`compute_output_shape()` (*langml.plm.layers.TokenEmbedding method*), 49

`compute_output_shape()` (*langml.plm.Masked method*), 51

`compute_output_shape()` (*langml.plm.TokenEmbedding method*), 51

`compute_output_shape()` (*langml.prompt.models.PartialEmbedding method*), 56

`compute_output_shape()` (*langml.prompt.models.ptuning.PartialEmbedding method*), 56

`compute_output_shape()`

- (*langml.transformer.FeedForward* method), 71  
 compute\_output\_shape()  
 (*langml.transformer.layers.FeedForward* method), 70  
 ConditionalLayerNormalization (class in *langml.layers*), 43  
 ConditionalLayerNormalization (class in *langml.layers.layers*), 40  
 Constraint (in module *langml.tensor\_typing*), 74  
 contrastive() (in module *langml.baselines.contrastive.cli*), 24  
 CRF (class in *langml.layers*), 41  
 CRF (class in *langml.layers.crf*), 37  
 crf\_binary\_score() (in module *langml.third\_party.crf*), 65  
 crf\_constrained\_decode() (in module *langml.third\_party.crf*), 68  
 crf\_decode() (in module *langml.third\_party.crf*), 68  
 crf\_decode\_backward() (in module *langml.third\_party.crf*), 68  
 crf\_decode\_forward() (in module *langml.third\_party.crf*), 67  
 crf\_filtered\_inputs() (in module *langml.third\_party.crf*), 63  
 crf\_forward() (in module *langml.third\_party.crf*), 65  
 crf\_log\_likelihood() (in module *langml.third\_party.crf*), 64  
 crf\_log\_norm() (in module *langml.third\_party.crf*), 64  
 crf\_multitag\_sequence\_score() (in module *langml.third\_party.crf*), 63  
 crf\_sequence\_score() (in module *langml.third\_party.crf*), 63  
 crf\_unary\_score() (in module *langml.third\_party.crf*), 65  
 CrfDecodeForwardRnnCell (class in *langml.third\_party.crf*), 67  
 custom\_objects (in module *langml.activations*), 72  
 custom\_objects (in module *langml.layers*), 46  
 custom\_objects (in module *langml.plm*), 52  
 custom\_objects (in module *langml.prompt.models*), 57  
 custom\_objects (in module *langml.transformer*), 71
- ## D
- DataGenerator (class in *langml.prompt.clf.ptuning*), 53  
 DataLoader (class in *langml.baselines.clf.data\_loader*), 20  
 DataLoader (class in *langml.baselines.contrastive.simcse*), 23  
 DataLoader (class in *langml.baselines.contrastive.simcse.data\_loader*), 22  
 DataLoader (class in *langml.baselines.matching.sbert*), 27  
 DataLoader (class in *langml.baselines.matching.sbert.data\_loader*), 26
- DataLoader (class in *langml.baselines.ner.data\_loader*), 30  
 debug (in module *langml.log*), 73  
 decode() (*langml.tokenizer.SPTokenizer* method), 77  
 decode() (*langml.tokenizer.Tokenizer* method), 77  
 decode() (*langml.tokenizer.WPTokenizer* method), 78  
 decode\_label() (*langml.prompt.base.Template* method), 57  
 decode\_label() (*langml.prompt.Template* method), 58  
 decode\_one() (*langml.baselines.ner.Infer* method), 31  
 deprecated\_warning() (in module *langml.utils*), 78
- ## E
- EmbeddingMatching (class in *langml.plm*), 51  
 EmbeddingMatching (class in *langml.plm.layers*), 49  
 EN\_PUNCTUATIONS (in module *langml.baselines.contrastive.utils*), 25  
 enable\_truncation() (*langml.tokenizer.Tokenizer* method), 75  
 encode() (*langml.tokenizer.Tokenizer* method), 76  
 encode\_batch() (*langml.tokenizer.Tokenizer* method), 76  
 encode\_data() (*langml.baselines.ner.data\_loader.DataLoader* method), 30  
 encode\_label\_tokens\_map() (*langml.prompt.base.Template* method), 57  
 encode\_label\_tokens\_map() (*langml.prompt.Template* method), 58  
 encode\_template() (*langml.prompt.base.Template* method), 57  
 encode\_template() (*langml.prompt.Template* method), 58  
 Encoding (class in *langml.tokenizer*), 75  
 end\_of\_chunk() (in module *langml.third\_party.conlleval*), 60  
 error (in module *langml.log*), 73  
 EvalCounts (class in *langml.third\_party.conlleval*), 60  
 evaluate() (in module *langml.third\_party.conlleval*), 60  
 export\_model\_v1() (in module *langml.model*), 74
- ## F
- FeedForward (class in *langml.transformer*), 71  
 FeedForward (class in *langml.transformer.layers*), 70  
 fit() (*langml.prompt.base.BasePromptTask* method), 57  
 fit() (*langml.prompt.clf.ptuning.PTuningForClassification* method), 53  
 fit() (*langml.prompt.clf.PTuningForClassification* method), 54  
 fit() (*langml.prompt.PTuningForClassification* method), 58  
 FormatError, 60

`from_config()` (*langml.third\_party.crf.CrfDecodeForward* class method), 67

## G

`GatedAttentionUnit` (class in *langml.layers*), 45

`GatedAttentionUnit` (class in *langml.layers.attention*), 36

`gelu()` (in module *langml.activations*), 72

`get_config()` (*langml.layers.AbsolutePositionEmbedding* method), 42

`get_config()` (*langml.layers.attention.GatedAttentionUnit* method), 37

`get_config()` (*langml.layers.attention.MultiHeadAttention* method), 36

`get_config()` (*langml.layers.attention.ScaledDotProductAttention* method), 35

`get_config()` (*langml.layers.attention.SelfAdditiveAttention* method), 35

`get_config()` (*langml.layers.attention.SelfAttention* method), 34

`get_config()` (*langml.layers.ConditionalLayerNormalization* method), 43

`get_config()` (*langml.layers.CRF* method), 41

`get_config()` (*langml.layers.crf.CRF* method), 38

`get_config()` (*langml.layers.GatedAttentionUnit* method), 46

`get_config()` (*langml.layers.layer\_norm.LayerNorm* method), 38

`get_config()` (*langml.layers.LayerNorm* method), 42

`get_config()` (*langml.layers.layers.AbsolutePositionEmbedding* method), 39

`get_config()` (*langml.layers.layers.ConditionalLayerNormalization* method), 40

`get_config()` (*langml.layers.layers.ScaleOffset* method), 40

`get_config()` (*langml.layers.layers.SineCosinePositionEmbedding* method), 39

`get_config()` (*langml.layers.MultiHeadAttention* method), 45

`get_config()` (*langml.layers.ScaledDotProductAttention* method), 44

`get_config()` (*langml.layers.ScaleOffset* method), 43

`get_config()` (*langml.layers.SelfAdditiveAttention* method), 44

`get_config()` (*langml.layers.SelfAttention* method), 43

`get_config()` (*langml.layers.SineCosinePositionEmbedding* method), 42

`get_config()` (*langml.plm.EmbeddingMatching* method), 51

`get_config()` (*langml.plm.layers.EmbeddingMatching* method), 49

`get_config()` (*langml.plm.layers.Masked* method), 50

`get_config()` (*langml.plm.Masked* method), 51

`get_config()` (*langml.third\_party.crf.CrfDecodeForwardRnnCell* method), 67

`get_config()` (*langml.transformer.FeedForward* method), 71

`get_config()` (*langml.transformer.layers.FeedForward* method), 70

`get_custom_objects()` (*langml.layers.AbsolutePositionEmbedding* static method), 42

`get_custom_objects()` (*langml.layers.attention.GatedAttentionUnit* static method), 37

`get_custom_objects()` (*langml.layers.attention.MultiHeadAttention* static method), 36

`get_custom_objects()` (*langml.layers.attention.ScaledDotProductAttention* static method), 36

`get_custom_objects()` (*langml.layers.attention.SelfAdditiveAttention* static method), 35

`get_custom_objects()` (*langml.layers.attention.SelfAttention* static method), 34

`get_custom_objects()` (*langml.layers.ConditionalLayerNormalization* static method), 43

`get_custom_objects()` (*langml.layers.CRF* static method), 41

`get_custom_objects()` (*langml.layers.crf.CRF* static method), 38

`get_custom_objects()` (*langml.layers.GatedAttentionUnit* static method), 46

`get_custom_objects()` (*langml.layers.layer\_norm.LayerNorm* static method), 39

`get_custom_objects()` (*langml.layers.LayerNorm* static method), 42

`get_custom_objects()` (*langml.layers.layers.AbsolutePositionEmbedding* static method), 39

`get_custom_objects()` (*langml.layers.layers.ConditionalLayerNormalization* static method), 40

`get_custom_objects()` (*langml.layers.layers.ScaleOffset* static method), 40

`get_custom_objects()` (*langml.layers.layers.SineCosinePositionEmbedding* static method), 39

`get_custom_objects()` (*langml.layers.MultiHeadAttention* static method), 45



[get\\_custom\\_objects\(\)](#)  
*(langml.layers.ScaledDotProductAttention static method)*, 45  
[get\\_custom\\_objects\(\)](#) *(langml.layers.ScaleOffset static method)*, 43  
[get\\_custom\\_objects\(\)](#)  
*(langml.layers.SelfAdditiveAttention static method)*, 44  
[get\\_custom\\_objects\(\)](#) *(langml.layers.SelfAttention static method)*, 44  
[get\\_custom\\_objects\(\)](#)  
*(langml.layers.SineCosinePositionEmbedding static method)*, 42  
[get\\_custom\\_objects\(\)](#)  
*(langml.plm.EmbeddingMatching static method)*, 51  
[get\\_custom\\_objects\(\)](#)  
*(langml.plm.layers.EmbeddingMatching static method)*, 49  
[get\\_custom\\_objects\(\)](#) *(langml.plm.layers.Masked static method)*, 49  
[get\\_custom\\_objects\(\)](#)  
*(langml.plm.layers.TokenEmbedding static method)*, 49  
[get\\_custom\\_objects\(\)](#) *(langml.plm.Masked static method)*, 51  
[get\\_custom\\_objects\(\)](#) *(langml.plm.TokenEmbedding static method)*, 51  
[get\\_custom\\_objects\(\)](#)  
*(langml.prompt.models.PartialEmbedding static method)*, 56  
[get\\_custom\\_objects\(\)](#)  
*(langml.prompt.models.ptuning.PartialEmbedding static method)*, 55  
[get\\_custom\\_objects\(\)](#)  
*(langml.transformer.FeedForward static method)*, 71  
[get\\_custom\\_objects\(\)](#)  
*(langml.transformer.layers.FeedForward static method)*, 70  
[get\\_embedding\(\)](#) *(langml.plm.bert.BERT method)*, 48  
[get\\_initial\\_state\(\)](#)  
*(langml.third\_party.crf.AbstractRNNCell method)*, 67  
[get\\_inputs\(\)](#) *(langml.plm.bert.BERT method)*, 48  
[get\\_pooling\\_output\(\)](#)  
*(langml.baselines.contrastive.simcse.model.SimCSE method)*, 23  
[get\\_pooling\\_output\(\)](#)  
*(langml.baselines.contrastive.simcse.SimCSE method)*, 24  
[get\\_pooling\\_output\(\)](#)  
*(langml.baselines.matching.sbert.model.SentenceBert method)*, 26  
[get\\_pooling\\_output\(\)](#)  
*(langml.baselines.matching.sbert.SentenceBert method)*, 27  
[get\\_random\\_string\(\)](#) *(in module langml.model)*, 74  
[get\\_vocab\(\)](#) *(langml.tokenizer.SPTokenizer method)*, 77  
[get\\_vocab\(\)](#) *(langml.tokenizer.Tokenizer method)*, 77  
[get\\_vocab\(\)](#) *(langml.tokenizer.WPTokenizer method)*, 78  
[get\\_vocab\\_size\(\)](#) *(langml.tokenizer.SPTokenizer method)*, 77  
[get\\_vocab\\_size\(\)](#) *(langml.tokenizer.Tokenizer method)*, 77  
[get\\_vocab\\_size\(\)](#) *(langml.tokenizer.WPTokenizer method)*, 77  
[get\\_weight\\_name\(\)](#) *(langml.plm.bert.BERT method)*, 48  
  
**I**  
[id\\_to\\_token\(\)](#) *(langml.tokenizer.SPTokenizer method)*, 77  
[id\\_to\\_token\(\)](#) *(langml.tokenizer.Tokenizer method)*, 77  
[id\\_to\\_token\(\)](#) *(langml.tokenizer.WPTokenizer method)*, 77  
[ids](#) *(langml.tokenizer.Encoding attribute)*, 75  
[Infer](#) *(class in langml.baselines.clf)*, 21  
[Infer](#) *(class in langml.baselines.ner)*, 31  
[info](#) *(in module langml.log)*, 73  
[Initializer](#) *(in module langml.tensor\_typing)*, 74  
[is\\_trainable\(\)](#) *(langml.plm.bert.BERT method)*, 48  
  
**L**  
[langml](#)  
module, 17  
[langml.activations](#)  
module, 72  
[langml.baselines](#)  
module, 17  
[langml.baselines.clf](#)  
module, 17  
[langml.baselines.clf.bert](#)  
module, 17  
[langml.baselines.clf.bilstm](#)  
module, 18  
[langml.baselines.clf.cli](#)  
module, 18  
[langml.baselines.clf.dataloader](#)  
module, 20  
[langml.baselines.clf.textcnn](#)  
module, 20  
[langml.baselines.cli](#)  
module, 32  
[langml.baselines.contrastive](#)  
module, 21

|                                                             |                                                                               |
|-------------------------------------------------------------|-------------------------------------------------------------------------------|
| langml.baselines.contrastive.cli<br>module, 24              | langml.plm.albert<br>module, 46                                               |
| langml.baselines.contrastive.simcse<br>module, 21           | langml.plm.bert<br>module, 47                                                 |
| langml.baselines.contrastive.simcse.dataloder<br>module, 21 | langml.plm.layers<br>module, 49                                               |
| langml.baselines.contrastive.simcse.model<br>module, 22     | langml.prompt<br>module, 53                                                   |
| langml.baselines.contrastive.utils<br>module, 25            | langml.prompt.base<br>module, 57                                              |
| langml.baselines.matching<br>module, 25                     | langml.prompt.clf<br>module, 53                                               |
| langml.baselines.matching.cli<br>module, 28                 | langml.prompt.clf.ptuning<br>module, 53                                       |
| langml.baselines.matching.sbert<br>module, 25               | langml.prompt.clf.utils<br>module, 54                                         |
| langml.baselines.matching.sbert.dataloder<br>module, 25     | langml.prompt.models<br>module, 55                                            |
| langml.baselines.matching.sbert.model<br>module, 26         | langml.prompt.models.ptuning<br>module, 55                                    |
| langml.baselines.ner<br>module, 28                          | langml.tensor_typing<br>module, 74                                            |
| langml.baselines.ner.bert_crf<br>module, 28                 | langml.third_party<br>module, 59                                              |
| langml.baselines.ner.cli<br>module, 29                      | langml.third_party.conlleval<br>module, 59                                    |
| langml.baselines.ner.dataloader<br>module, 30               | langml.third_party.crf<br>module, 60                                          |
| langml.baselines.ner.lstm_crf<br>module, 30                 | langml.tokenizer<br>module, 74                                                |
| langml.cli<br>module, 72                                    | langml.transformer<br>module, 69                                              |
| langml.common<br>module, 33                                 | langml.transformer.encoder<br>module, 69                                      |
| langml.common.evaluator<br>module, 33                       | langml.transformer.layers<br>module, 70                                       |
| langml.common.evaluator.spearman<br>module, 33              | langml.utils<br>module, 78                                                    |
| langml.layers<br>module, 33                                 | LayerNorm (class in langml.layers), 41                                        |
| langml.layers.attention<br>module, 33                       | LayerNorm (class in langml.layers.layer_norm), 38                             |
| langml.layers.crf<br>module, 37                             | load() (langml.prompt.clf.ptuning.PTuningForClassification<br>method), 53     |
| langml.layers.layer_norm<br>module, 38                      | load() (langml.prompt.clf.PTuningForClassification<br>method), 55             |
| langml.layers.layers<br>module, 39                          | load() (langml.prompt.PTuningForClassification<br>method), 58                 |
| langml.log<br>module, 73                                    | load_albert() (in module langml.plm), 52                                      |
| langml.model<br>module, 73                                  | load_albert() (in module langml.plm.albert), 47                               |
| langml.plm<br>module, 46                                    | load_bert() (in module langml.plm), 51                                        |
|                                                             | load_bert() (in module langml.plm.bert), 48                                   |
|                                                             | load_data() (langml.baselines.BaseDataLoader static<br>method), 32            |
|                                                             | load_data() (langml.baselines.clf.dataloader.DataLoader<br>static method), 20 |

[load\\_data\(\)](#) (*langml.baselines.contrastive.simcse.DataLoader* static method), 23  
[load\\_data\(\)](#) (*langml.baselines.contrastive.simcse.dataloader.DataLoader* static method), 22  
[load\\_data\(\)](#) (*langml.baselines.matching.sbert.DataLoader* static method), 27  
[load\\_data\(\)](#) (*langml.baselines.matching.sbert.dataloader.DataLoader* static method), 26  
[load\\_data\(\)](#) (*langml.baselines.ner.dataloader.DataLoader* static method), 30  
[load\\_frozen\(\)](#) (in module *langml.model*), 74  
[load\\_variables\(\)](#) (in module *langml.utils*), 78  
[loss](#) (*langml.layers.CRF* property), 41  
[loss](#) (*langml.layers.crf.CRF* property), 38  
[lstm\\_crf\(\)](#) (in module *langml.baselines.ner.cli*), 29  
[LSTMCRF](#) (class in *langml.baselines.ner.lstm\_crf*), 30

## M

[main\(\)](#) (in module *langml.cli*), 72  
[main\(\)](#) (in module *langml.third\_party.conlleval*), 60  
[make\\_iter\(\)](#) (*langml.baselines.BaseDataLoader* method), 32  
[make\\_iter\(\)](#) (*langml.baselines.clf.dataloader.DataLoader* method), 20  
[make\\_iter\(\)](#) (*langml.baselines.clf.dataloader.TFDataLoader* method), 20  
[make\\_iter\(\)](#) (*langml.baselines.contrastive.simcse.DataLoader* method), 23  
[make\\_iter\(\)](#) (*langml.baselines.contrastive.simcse.dataloader.DataLoader* method), 22  
[make\\_iter\(\)](#) (*langml.baselines.contrastive.simcse.dataloader.TFDataLoader* method), 22  
[make\\_iter\(\)](#) (*langml.baselines.contrastive.simcse.TFDataLoader* method), 24  
[make\\_iter\(\)](#) (*langml.baselines.matching.sbert.DataLoader* method), 27  
[make\\_iter\(\)](#) (*langml.baselines.matching.sbert.dataloader.DataLoader* method), 26  
[make\\_iter\(\)](#) (*langml.baselines.matching.sbert.dataloader.TFDataLoader* method), 26  
[make\\_iter\(\)](#) (*langml.baselines.matching.sbert.TFDataLoader* method), 27  
[make\\_iter\(\)](#) (*langml.baselines.ner.dataloader.DataLoader* method), 30  
[make\\_iter\(\)](#) (*langml.baselines.ner.dataloader.TFDataLoader* method), 30  
[make\\_iter\(\)](#) (*langml.prompt.base.BaseDataGenerator* method), 58  
[make\\_iter\(\)](#) (*langml.prompt.clf.ptuning.DataGenerator* method), 53  
[MASK](#) (*langml.tokenizer.SpecialTokens* attribute), 75  
[Masked](#) (class in *langml.plm*), 51  
[Masked](#) (class in *langml.plm.layers*), 49

[matching\(\)](#) (in module *langml.baselines.matching.cli*), 28  
[model\\_generator\\_template\\_tokens\(\)](#) (in module *langml.prompt.clf.utils*), 54  
[Metrics](#) (in module *langml.third\_party.conlleval*), 60  
[metrics\(\)](#) (in module *langml.third\_party.conlleval*), 60  
[MultiTaskCallback](#) (class in *langml.prompt.clf.utils*), 54  
[Models](#) (in module *langml.baselines.clf*), 21  
[Models](#) (in module *langml.baselines.ner*), 31  
[Models](#) (in module *langml.tensor\_typing*), 74  
[module](#)  
[langml](#), 17  
[langml.activations](#), 72  
[langml.baselines](#), 17  
[langml.baselines.clf](#), 17  
[langml.baselines.clf.bert](#), 17  
[langml.baselines.clf.bilstm](#), 18  
[langml.baselines.clf.cli](#), 18  
[langml.baselines.clf.dataloader](#), 20  
[langml.baselines.clf.textcnn](#), 20  
[langml.baselines.cli](#), 32  
[langml.baselines.contrastive](#), 21  
[langml.baselines.contrastive.cli](#), 24  
[langml.baselines.contrastive.simcse](#), 21  
[langml.baselines.contrastive.simcse.dataloader](#), 21  
[langml.baselines.contrastive.simcse.model](#), 22  
[langml.baselines.contrastive.utils](#), 25  
[langml.baselines.matching](#), 25  
[langml.baselines.matching.cli](#), 28  
[langml.baselines.matching.sbert](#), 25  
[langml.baselines.matching.sbert.dataloader](#), 25  
[langml.baselines.matching.sbert.model](#), 26  
[langml.baselines.ner](#), 28  
[langml.baselines.ner.bert\\_crf](#), 28  
[langml.baselines.ner.cli](#), 29  
[langml.baselines.ner.dataloader](#), 30  
[langml.baselines.ner.lstm\\_crf](#), 30  
[langml.cli](#), 72  
[langml.common](#), 33  
[langml.common.evaluator](#), 33  
[langml.common.evaluator.spearman](#), 33  
[langml.layers](#), 33  
[langml.layers.attention](#), 33  
[langml.layers.crf](#), 37  
[langml.layers.layer\\_norm](#), 38  
[langml.layers.layers](#), 39  
[langml.log](#), 73  
[langml.model](#), 73  
[langml.plm](#), 46  
[langml.plm.albert](#), 46  
[langml.plm.bert](#), 47

- langml.plm.layers, 49
  - langml.prompt, 53
  - langml.prompt.base, 57
  - langml.prompt.clf, 53
  - langml.prompt.clf.ptuning, 53
  - langml.prompt.clf.utils, 54
  - langml.prompt.models, 55
  - langml.prompt.models.ptuning, 55
  - langml.tensor\_typing, 74
  - langml.third\_party, 59
  - langml.third\_party.conlleval, 59
  - langml.third\_party.crf, 60
  - langml.tokenizer, 74
  - langml.transformer, 69
  - langml.transformer.encoder, 69
  - langml.transformer.layers, 70
  - langml.utils, 78
  - MultiHeadAttention (class in langml.layers), 45
  - MultiHeadAttention (class in langml.layers.attention), 36
- ## N
- ner() (in module langml.baselines.ner.cli), 29
  - Number (in module langml.tensor\_typing), 74
- ## O
- on\_epoch\_end() (langml.prompt.clf.utils.MetricsCallback method), 54
  - on\_train\_begin() (langml.prompt.clf.utils.MetricsCallback method), 54
  - on\_train\_end() (langml.prompt.clf.utils.MetricsCallback method), 54
  - Optimizer (in module langml.tensor\_typing), 74
  - output\_size (langml.third\_party.crf.AbstractRNNCell property), 67
  - output\_size (langml.third\_party.crf.CrfDecodeForwardRNNCell property), 67
- ## P
- PAD (langml.tokenizer.SpecialTokens attribute), 75
  - Parameters (class in langml.baselines), 32
  - parse\_args() (in module langml.third\_party.conlleval), 60
  - parse\_tag() (in module langml.third\_party.conlleval), 60
  - PartialEmbedding (class in langml.prompt.models), 56
  - PartialEmbedding (class in langml.prompt.models.ptuning), 55
  - predict() (langml.prompt.base.BasePromptTask method), 57
  - predict() (langml.prompt.clf.ptuning.PTuningForClassification method), 53
  - predict() (langml.prompt.clf.PTuningForClassification method), 55
  - predict() (langml.prompt.PTuningForClassification method), 58
  - print\_log() (in module langml.log), 73
  - PTuningForClassification (class in langml.prompt), 58
  - PTuningForClassification (class in langml.prompt.clf), 54
  - PTuningForClassification (class in langml.prompt.clf.ptuning), 53
  - PTuningPrompt (class in langml.prompt), 58
  - PTuningPrompt (class in langml.prompt.models), 57
  - PTuningPrompt (class in langml.prompt.models.ptuning), 56
- ## R
- raw\_tokenizer() (langml.tokenizer.Tokenizer method), 76
  - re\_split (in module langml.baselines.ner), 31
  - Regularizer (in module langml.tensor\_typing), 74
  - relu2() (in module langml.activations), 72
  - report() (in module langml.third\_party.conlleval), 60
  - report\_notprint() (in module langml.third\_party.conlleval), 60
  - return\_report() (in module langml.third\_party.conlleval), 60
- ## S
- save\_frozen() (in module langml.model), 74
  - SAVED\_MODEL\_TAG (in module langml.model), 74
  - sbert() (in module langml.baselines.matching.cli), 28
  - ScaledDotProductAttention (class in langml.layers), 44
  - ScaledDotProductAttention (class in langml.layers.attention), 35
  - ScaleOffset (class in langml.layers), 43
  - ScaleOffset (class in langml.layers.layers), 40
  - segment\_ids (langml.tokenizer.Encoding attribute), 75
  - SelfAdditiveAttention (class in langml.layers), 44
  - SelfAdditiveAttention (class in langml.layers.attention), 34
  - SelfAttention (class in langml.layers), 43
  - SelfAttention (class in langml.layers.attention), 34
  - SentenceBert (class in langml.baselines.matching.sbert), 27
  - SentenceBert (class in langml.baselines.matching.sbert.model), 26
  - SEP (langml.tokenizer.SpecialTokens attribute), 75
  - sequence\_lower() (langml.tokenizer.Tokenizer method), 76
  - sequence\_truncating() (langml.tokenizer.Tokenizer method), 76
  - SimCSE (class in langml.baselines.contrastive.simcse), 24



SimCSE (class in langml.baselines.contrastive.simcse.model), 23  
 token\_to\_id() (langml.tokenizer.WPTokenizer method), 77  
 simcse() (in module langml.baselines.contrastive.cli), 24  
 simcse\_loss() (in module langml.baselines.contrastive.simcse.model), 23  
 SineCosinePositionEmbedding (class in langml.layers), 42  
 SineCosinePositionEmbedding (class in langml.layers.layers), 39  
 SpearmanEvaluator (class in langml.common.evaluator), 33  
 SpearmanEvaluator (class in langml.common.evaluator.spearman), 33  
 SpecialTokens (class in langml.tokenizer), 75  
 SPTokenizer (class in langml.tokenizer), 77  
 start\_of\_chunk() (in module langml.third\_party.conlleval), 60  
 state\_size (langml.third\_party.crf.AbstractRNNCell property), 67  
 state\_size (langml.third\_party.crf.CrfDecodeForwardRnnCell property), 67  
 stem() (langml.tokenizer.Tokenizer method), 76  
**T**  
 Template (class in langml.prompt), 58  
 Template (class in langml.prompt.base), 57  
 Tensors (in module langml.tensor\_typing), 74  
 textcnn() (in module langml.baselines.clf.cli), 19  
 TextCNNClassifier (class in langml.baselines.clf.textcnn), 20  
 TF\_KERAS (in module langml), 79  
 TF\_KERAS (in module langml.layers), 41  
 TF\_KERAS (in module langml.transformer), 71  
 TF\_VERSION (in module langml), 79  
 TF\_VERSION (in module langml.baselines.clf), 21  
 TF\_VERSION (in module langml.baselines.ner), 31  
 TFDataLoader (class in langml.baselines.clf.dataloader), 20  
 TFDataLoader (class in langml.baselines.contrastive.simcse), 23  
 TFDataLoader (class in langml.baselines.contrastive.simcse.dataloder), 22  
 TFDataLoader (class in langml.baselines.matching.sbert), 27  
 TFDataLoader (class in langml.baselines.matching.sbert.dataloder), 26  
 TFDataLoader (class in langml.baselines.ner.dataloader), 30  
 token\_to\_id() (langml.tokenizer.SPTokenizer method), 77  
 token\_to\_id() (langml.tokenizer.Tokenizer method), 77  
 TokenEmbedding (class in langml.plm), 51  
 TokenEmbedding (class in langml.plm.layers), 49  
 tokenize() (langml.tokenizer.SPTokenizer method), 77  
 tokenize() (langml.tokenizer.Tokenizer method), 76  
 tokenize() (langml.tokenizer.WPTokenizer method), 78  
 Tokenizer (class in langml.tokenizer), 75  
 tokens (langml.tokenizer.Encoding attribute), 75  
 tokens() (langml.tokenizer.SpecialTokens method), 75  
 tokens\_mapping() (langml.tokenizer.Tokenizer method), 75  
 train() (in module langml.baselines.clf.cli), 19  
 train() (in module langml.baselines.ner.cli), 29  
 trans (langml.layers.CRF property), 41  
 trans (langml.layers.crf.CRF property), 38  
 TransformerEncoder (class in langml.transformer.encoder), 69  
 TransformerEncoderBlock (class in langml.transformer.encoder), 69  
**U**  
 uniq() (in module langml.third\_party.conlleval), 60  
 UNK (langml.tokenizer.SpecialTokens attribute), 75  
**V**  
 viterbi\_decode() (in module langml.third\_party.crf), 63  
**W**  
 warn (in module langml.log), 73  
 whitespace\_tokenize() (in module langml.baselines.contrastive.utils), 25  
 WPTokenizer (class in langml.tokenizer), 77