nlpaug Documentation

Release 1.1.11

Edward Ma

Jul 07, 2022

Contents:

nlpgaug is a library for textual augmentation in machine learning experiments. The goal is improving deep learning model performance by generating textual data. It also able to generate adversarial examples to prevent adversarial attacks.

CHAPTER 1

Overview

This python library helps you with augmenting nlp for your machine learning projects. Visit this introduction to understand about Data Augmentation in NLP. Augmenter is the basic element of augmentation while Flow is a pipeline to orchestra multi augmenter together.

- Data Augmentation library for Text
- Data Augmentation library for Speech Recognition
- Data Augmentation library for Audio
- Does your NLP model able to prevent adversarial attack?

CHAPTER 2

Example

The following examples show a standard use case for augmenter.

- Audio augmenters
- Textual augmenters
- Spectrogram augmenters
- Custom augmenter
- TF-IDF model training
- Flow

CHAPTER 3

Augmenter

3.1 Audio Augmenter

3.1.1 nlpaug.augmenter.audio.crop

Augmenter that apply cropping operation to audio.

class nlpaug.augmenter.audio.crop.**CropAug**(*sampling_rate=None*, *zone=(0.2, 0.8)*, *coverage=0.1*, *duration=None*, *name='Crop_Aug'*, *verbose=0*, *stateless=True*)

Bases: nlpaug.augmenter.audio.audio_augmenter.AudioAugmenter

Parameters

- **sampling_rate** (*int*) – Sampling rate of input audio. Mandatory if duration is provided.
- **zone** (*tuple*) – Assign a zone for augmentation. Default value is (0.2, 0.8) which means that no any augmentation will be applied in first 20% and last 20% of whole audio.
- **coverage** (*float*) – Portion of augmentation. Value should be between 0 and 1. If *0.1* is assigned, augment operation will be applied to target audio segment. For example, the audio duration is 60 seconds while zone and coverage are (0.2, 0.8) and 0.7 respectively. 25.2 seconds ((0.8-0.2)*0.7*60) audio will be augmented.
- **duration** (*int*) – Duration of augmentation (in second). Default value is None. If value is provided. *coverage* value will be ignored.
- **name** (*str*) – Name of this augmenter

```
>>> import nlpaug.augmenter.audio as naa
>>> aug = naa.CropAug(sampling_rate=44010)
```

augment(*data*, *n=1*, *num_thread=1*)

Parameters

- **data** (*object/list*) – Data for augmentation. It can be list of data (e.g. list of string or numpy) or single element (e.g. string or numpy). Numpy format only supports audio or spectrogram data. For text data, only support string or list of string.
- **n** (*int*) – Default is 1. Number of unique augmented output. Will be force to 1 if input is list of data
- **num_thread** (*int*) – Number of thread for data augmentation. Use this option when you are using CPU and n is larger than 1

Returns Augmented data

```
>>> augmented_data = aug.augment(data)
```

3.1.2 nlpaug.augmenter.audio.loudness

Augmenter that apply adjusting loudness operation to audio.

class nlpaug.augmenter.audio.loudness.**LoudnessAug**(*zone=(0.2, 0.8), coverage=1.0, factor=(0.5, 2), name='Loudness_Aug', verbose=0, stateless=True*)

Bases: nlpaug.augmenter.audio.audio_augmenter.AudioAugmenter

Parameters

- **zone** (*tuple*) – Assign a zone for augmentation. Default value is (0.2, 0.8) which means that no any augmentation will be applied in first 20% and last 20% of whole audio.
- **coverage** (*float*) – Portion of augmentation. Value should be between 0 and 1. If *1* is assigned, augment operation will be applied to target audio segment. For example, the audio duration is 60 seconds while zone and coverage are (0.2, 0.8) and 0.7 respectively. 42 seconds ((0.8-0.2)*0.7*60) audio will be augmented.
- **factor** (*tuple*) – Input data volume will be increased (decreased). Augmented value will be picked within the range of this tuple value. Volume will be reduced if value is between 0 and 1.
- **name** (*str*) – Name of this augmenter

```
>>> import nlpaug.augmenter.audio as naa
>>> aug = naa.LoudnessAug()
```

augment(*data*, *n=1*, *num_thread=1*)

Parameters

- **data** (*object/list*) – Data for augmentation. It can be list of data (e.g. list of string or numpy) or single element (e.g. string or numpy). Numpy format only supports audio or spectrogram data. For text data, only support string or list of string.
- **n** (*int*) – Default is 1. Number of unique augmented output. Will be force to 1 if input is list of data
- **num_thread** (*int*) – Number of thread for data augmentation. Use this option when you are using CPU and n is larger than 1

Returns Augmented data

```
>>> augmented_data = aug.augment(data)
```

3.1.3 nlpaug.augmenter.audio.mask

Augmenter that apply mask operation to audio.

class nlpaug.augmenter.audio.mask.**MaskAug**(*sampling_rate=None*, *zone=(0.2, 0.8)*, *coverage=1.0*, *duration=None*, *mask_with_noise=True*, *name='Mask_Aug'*, *verbose=0*, *stateless=True*)

Bases: nlpaug.augmenter.audio.audio_augmenter.AudioAugmenter

Parameters

- **sampling_rate** (*int*) – Sampling rate of input audio. Mandatory if duration is provided.
- **zone** (*tuple*) – Assign a zone for augmentation. Default value is (0.2, 0.8) which means that no any augmentation will be applied in first 20% and last 20% of whole audio.
- **coverage** (*float*) – Portion of augmentation. Value should be between 0 and 1. If *1* is assigned, augment operation will be applied to target audio segment. For example, the audio duration is 60 seconds while zone and coverage are (0.2, 0.8) and 0.7 respectively. 42 seconds ((0.8-0.2)*0.7*60) audio will be augmented.
- **duration** (*int*) – Duration of augmentation (in second). Default value is None. If value is provided. *coverage* value will be ignored.
- **mask_with_noise** (*bool*) – If it is True, targeting area will be replaced by noise. Otherwise, it will be replaced by 0.
- **name** (*str*) – Name of this augmenter

```
>>> import nlpaug.augmenter.audio as naa
>>> aug = naa.MaskAug(sampling_rate=44010)
```

augment(*data*, *n=1*, *num_thread=1*)

Parameters

- **data** (*object/list*) – Data for augmentation. It can be list of data (e.g. list of string or numpy) or single element (e.g. string or numpy). Numpy format only supports audio or spectrogram data. For text data, only support string or list of string.
- **n** (*int*) – Default is 1. Number of unique augmented output. Will be force to 1 if input is list of data
- **num_thread** (*int*) – Number of thread for data augmentation. Use this option when you are using CPU and n is larger than 1

Returns Augmented data

```
>>> augmented_data = aug.augment(data)
```

3.1.4 nlpaug.augmenter.audio.noise

Augmenter that apply noise injection operation to audio.

class nlpaug.augmenter.audio.noise.**NoiseAug**(*zone=(0.2, 0.8)*, *coverage=1.0*, *color='white'*, *noises=None*, *name='Noise_Aug'*, *verbose=0*, *stateless=True*)

Bases: nlpaug.augmenter.audio.audio_augmenter.AudioAugmenter

Parameters

- **zone** (*tuple*) – Assign a zone for augmentation. Default value is (0.2, 0.8) which means that no any augmentation will be applied in first 20% and last 20% of whole audio.
- **coverage** (*float*) – Portion of augmentation. Value should be between 0 and 1. If *1* is assigned, augment operation will be applied to target audio segment. For example, the audio duration is 60 seconds while zone and coverage are (0.2, 0.8) and 0.7 respectively. 42 seconds ((0.8-0.2)*0.7*60) audio will be augmented.
- **color** (*str*) – Colors of noise. Supported 'white', 'pink', 'red', 'brown', 'brownian', 'blue', 'azure', 'violet', 'purple' and 'random'. If 'random' is used, noise color will be picked randomly in each augment.
- **noises** (*list*) – Background noises for noise injection. You can provide more than one background noise and noise will be picked randomly. Expected format is list of numpy array. If this value is provided. *color* value will be ignored
- **name** (*str*) – Name of this augmenter

```
>>> import nlpaug.augmenter.audio as naa
>>> aug = naa.NoiseAug()
```

augment(*data*, *n=1*, *num_thread=1*)

Parameters

- **data** (*object/list*) – Data for augmentation. It can be list of data (e.g. list of string or numpy) or single element (e.g. string or numpy). Numpy format only supports audio or spectrogram data. For text data, only support string or list of string.
- **n** (*int*) – Default is 1. Number of unique augmented output. Will be force to 1 if input is list of data
- **num_thread** (*int*) – Number of thread for data augmentation. Use this option when you are using CPU and n is larger than 1

Returns Augmented data

```
>>> augmented_data = aug.augment(data)
```

3.1.5 nlpaug.augmenter.audio.normalization

Augmenter that apply mask normalization to audio.

class nlpaug.augmenter.audio.normalization.**NormalizeAug**(*method='max'*, *zone=(0.2, 0.8)*, *coverage=0.3*, *name='Normalize_Aug'*, *verbose=0*, *stateless=True*)

Bases: nlpaug.augmenter.audio.audio_augmenter.AudioAugmenter

Parameters

- **method** (*str*) – It supports 'minmax', 'max' and 'standard'. For 'minmax', data will be substracted by min value in data and dividing by range of max value and min value. For 'max', data will be divided by max value only. For 'standard', data will be substracted by mean value and dividing by value of standard deviation. If 'random' is used, method will be picked randomly in each augment.

- **zone** (*tuple*) – Assign a zone for augmentation. Default value is (0.2, 0.8) which means that no any augmentation will be applied in first 20% and last 20% of whole audio.
- **coverage** (*float*) – Portion of augmentation. Value should be between 0 and 1. If *0.1* is assigned, augment operation will be applied to target audio segment. For example, the audio duration is 60 seconds while zone and coverage are (0.2, 0.8) and 0.7 respectively. 25.2 seconds ((0.8-0.2)*0.7*60) audio will be augmented.
- **name** (*str*) – Name of this augmenter

```
>>> import nlpaug.augmenter.audio as naa
>>> aug = naa.NormalizeAug()
```

augment(*data*, *n=1*, *num_thread=1*)

Parameters

- **data** (*object/list*) – Data for augmentation. It can be list of data (e.g. list of string or numpy) or single element (e.g. string or numpy). Numpy format only supports audio or spectrogram data. For text data, only support string or list of string.
- **n** (*int*) – Default is 1. Number of unique augmented output. Will be force to 1 if input is list of data
- **num_thread** (*int*) – Number of thread for data augmentation. Use this option when you are using CPU and n is larger than 1

Returns Augmented data

```
>>> augmented_data = aug.augment(data)
```

3.1.6 nlpaug.augmenter.audio.pitch

Augmenter that apply pitch adjustment operation to audio.

class nlpaug.augmenter.audio.pitch.**PitchAug**(*sampling_rate*, *zone=(0.2, 0.8)*, *coverage=1.0*, *duration=None*, *factor=(-10, 10)*, *name='Pitch_Aug'*, *verbose=0*, *stateless=True*)

Bases: nlpaug.augmenter.audio.audio_augmenter.AudioAugmenter

Parameters

- **sampling_rate** (*int*) – Sampling rate of input audio.
- **zone** (*tuple*) – Assign a zone for augmentation. Default value is (0.2, 0.8) which means that no any augmentation will be applied in first 20% and last 20% of whole audio.
- **coverage** (*float*) – Portion of augmentation. Value should be between 0 and 1. If *1* is assigned, augment operation will be applied to target audio segment. For example, the audio duration is 60 seconds while zone and coverage are (0.2, 0.8) and 0.7 respectively. 42 seconds ((0.8-0.2)*0.7*60) audio will be augmented.
- **duration** (*int*) – Duration of augmentation (in second). Default value is None. If value is provided. *coverage* value will be ignored.
- **factor** (*tuple*) – Input data pitch will be increased (decreased). Augmented value will be picked within the range of this tuple value. Pitch will be reduced if value is between 0 and 1.
- **name** (*str*) – Name of this augmenter

```
>>> import nlpaug.augmenter.audio as naa
>>> aug = naa.PitchAug(sampling_rate=44010)
```

augment(*data*, *n=1*, *num_thread=1*)

Parameters

- **data** (*object/list*) – Data for augmentation. It can be list of data (e.g. list of string or numpy) or single element (e.g. string or numpy). Numpy format only supports audio or spectrogram data. For text data, only support string or list of string.
- **n** (*int*) – Default is 1. Number of unique augmented output. Will be force to 1 if input is list of data
- **num_thread** (*int*) – Number of thread for data augmentation. Use this option when you are using CPU and n is larger than 1

Returns Augmented data

```
>>> augmented_data = aug.augment(data)
```

3.1.7 nlpaug.augmenter.audio.shift

Augmenter that apply shifting operation to audio.

class nlpaug.augmenter.audio.shift.**ShiftAug**(*sampling_rate*, *duration=3*, *direction='random'*, *shift_direction='random'*, *name='Shift_Aug'*, *verbose=0*, *stateless=True*)

Bases: nlpaug.augmenter.audio.audio_augmenter.AudioAugmenter

Parameters

- **sampling_rate** (*int*) – Sampling rate of input audio.
- **duration** (*float*) – Max shifting segment (in second)
- **direction** (*str*) – Shifting segment to left, right or one of them. Value can be 'left', 'right' or 'random'
- **name** (*str*) – Name of this augmenter

```
>>> import nlpaug.augmenter.audio as naa
>>> aug = naa.ShiftAug(sampling_rate=44010)
```

augment(*data*, *n=1*, *num_thread=1*)

Parameters

- **data** (*object/list*) – Data for augmentation. It can be list of data (e.g. list of string or numpy) or single element (e.g. string or numpy). Numpy format only supports audio or spectrogram data. For text data, only support string or list of string.
- **n** (*int*) – Default is 1. Number of unique augmented output. Will be force to 1 if input is list of data
- **num_thread** (*int*) – Number of thread for data augmentation. Use this option when you are using CPU and n is larger than 1

Returns Augmented data

```
>>> augmented_data = aug.augment(data)
```

3.1.8 nlpaug.augmenter.audio.speed

Augmenter that apply speed adjustment operation to audio.

class nlpaug.augmenter.audio.speed.**SpeedAug**(*zone=(0.2, 0.8)*, *coverage=1.0*, *factor=(0.5, 2)*, *name='Speed_Aug'*, *verbose=0*, *stateless=True*)

Bases: nlpaug.augmenter.audio.audio_augmenter.AudioAugmenter

Parameters

- **zone** (*tuple*) – Assign a zone for augmentation. Default value is (0.2, 0.8) which means that no any augmentation will be applied in first 20% and last 20% of whole audio.
- **coverage** (*float*) – Portion of augmentation. Value should be between 0 and 1. If *1* is assigned, augment operation will be applied to target audio segment. For example, the audio duration is 60 seconds while zone and coverage are (0.2, 0.8) and 0.7 respectively. 42 seconds ((0.8-0.2)*0.7*60) audio will be augmented.
- **factor** (*tuple*) – Input data speed will be increased (decreased). Augmented value will be picked within the range of this tuple value. Speed will be reduced if value is between 0 and 1.
- **speed_range** (*tuple*) – Deprecated. Use *factor* indeed
- **name** (*str*) – Name of this augmenter

```
>>> import nlpaug.augmenter.audio as naa
>>> aug = naa.ShiftAug()
```

augment(*data*, *n=1*, *num_thread=1*)

Parameters

- **data** (*object/list*) – Data for augmentation. It can be list of data (e.g. list of string or numpy) or single element (e.g. string or numpy). Numpy format only supports audio or spectrogram data. For text data, only support string or list of string.
- **n** (*int*) – Default is 1. Number of unique augmented output. Will be force to 1 if input is list of data
- **num_thread** (*int*) – Number of thread for data augmentation. Use this option when you are using CPU and n is larger than 1

Returns Augmented data

```
>>> augmented_data = aug.augment(data)
```

3.1.9 nlpaug.augmenter.audio.vtlp

Augmenter that apply vocal tract length perturbation (VTLP) operation to audio.

class nlpaug.augmenter.audio.vtlp.**VtlpAug**(*sampling_rate*, *zone=(0.2, 0.8)*, *coverage=0.1*, *fhi=4800*, *factor=(0.9, 1.1)*, *name='Vtlp_Aug'*, *verbose=0*, *stateless=True*)

Bases: nlpaug.augmenter.audio.audio_augmenter.AudioAugmenter

Parameters

- **zone** (*tuple*) – Assign a zone for augmentation. Default value is (0.2, 0.8) which means that no any augmentation will be applied in first 20% and last 20% of whole audio.
- **coverage** (*float*) – Portion of augmentation. Value should be between 0 and 1. If *1* is assigned, augment operation will be applied to target audio segment. For example, the audio duration is 60 seconds while zone and coverage are (0.2, 0.8) and 0.7 respectively. 42 seconds ((0.8-0.2)*0.7*60) audio will be augmented.
- **factor** (*tuple*) – Input data vocal will be increased (decreased). Augmented value will be picked within the range of this tuple value. Vocal will be reduced if value is between 0 and 1.
- **fhi** (*int*) – Boundary frequency. Default value is 4800.
- **name** (*str*) – Name of this augmenter

```
>>> import nlpaug.augmenter.audio as naa
>>> aug = naa.VtlpAug()
```

augment (*data*, *n=1*, *num_thread=1*)

Parameters

- **data** (*object/list*) – Data for augmentation. It can be list of data (e.g. list of string or numpy) or single element (e.g. string or numpy). Numpy format only supports audio or spectrogram data. For text data, only support string or list of string.
- **n** (*int*) – Default is 1. Number of unique augmented output. Will be force to 1 if input is list of data
- **num_thread** (*int*) – Number of thread for data augmentation. Use this option when you are using CPU and n is larger than 1

Returns Augmented data

```
>>> augmented_data = aug.augment(data)
```

3.2 Character Augmenter

3.2.1 nlpaug.augmenter.char.keyboard

Augmenter that apply typo error simulation to textual input.

class nlpaug.augmenter.char.keyboard.**KeyboardAug** (*name='Keyboard_Aug'*, *aug_char_min=1*, *aug_char_max=10*, *aug_char_p=0.3*, *aug_word_p=0.3*, *aug_word_min=1*, *aug_word_max=10*, *stopwords=None*, *tokenizer=None*, *reverse_tokenizer=None*, *include_special_char=True*, *include_numeric=True*, *include_upper_case=True*, *lang='en'*, *verbose=0*, *stopwords_regex=None*, *model_path=None*, *min_char=4*)

Bases: nlpaug.augmenter.char.char_augmenter.CharAugmenter

Augmenter that simulate typo error by random values. For example, people may type i as o incorrectly. One keyboard distance is leveraged to replace character by possible keyboard error.

Parameters

- **aug_char_p** (*float*) – Percentage of character (per token) will be augmented.
- **aug_char_min** (*int*) – Minimum number of character will be augmented.
- **aug_char_max** (*int*) – Maximum number of character will be augmented. If None is passed, number of augmentation is calculated via aup_char_p. If calculated result from aug_char_p is smaller than aug_char_max, will use calculated result from aup_char_p. Otherwise, using aug_max.
- **aug_word_p** (*float*) – Percentage of word will be augmented.
- **aug_word_min** (*int*) – Minimum number of word will be augmented.
- **aug_word_max** (*int*) – Maximum number of word will be augmented. If None is passed, number of augmentation is calculated via aup_word_p. If calculated result from aug_word_p is smaller than aug_word_max, will use calculated result from aug_word_p. Otherwise, using aug_max.
- **stopwords** (*list*) – List of words which will be skipped from augment operation.
- **stopwords_regex** (*str*) – Regular expression for matching words which will be skipped from augment operation.
- **tokenizer** (*func*) – Customize tokenization process
- **reverse_tokenizer** (*func*) – Customize reverse of tokenization process
- **include_special_char** (*bool*) – Include special character
- **include_upper_case** (*bool*) – If True, upper case character may be included in augmented data.
- **include_numeric** (*bool*) – If True, numeric character may be included in augmented data.
- **min_char** (*int*) – If word less than this value, do not draw word for augmentation
- **model_path** (*str*) – Loading customize model from file system
- **lang** (*str*) – Indicate built-in language model. Default value is 'en'. Possible values are 'en', 'th' (Thai), 'tr'(Turkish), 'de'(German), 'es'(Spanish), 'fr'(French), 'it'(Italian), 'nl'(Dutch), 'pl'(Polish), 'uk'(Ukrainian), 'he'(Hebrew). If custom model is used (passing model_path), this value will be ignored.
- **name** (*str*) – Name of this augmenter

```
>>> import nlpaug.augmenter.char as nac
>>> aug = nac.KeyboardAug()
```

augment(*data*, *n=1*, *num_thread=1*)

Parameters

- **data** (*object/list*) – Data for augmentation. It can be list of data (e.g. list of string or numpy) or single element (e.g. string or numpy). Numpy format only supports audio or spectrogram data. For text data, only support string or list of string.

- **n** (*int*) – Default is 1. Number of unique augmented output. Will be force to 1 if input is list of data
- **num_thread** (*int*) – Number of thread for data augmentation. Use this option when you are using CPU and n is larger than 1

Returns Augmented data

```
>>> augmented_data = aug.augment(data)
```

3.2.2 nlpaug.augmenter.char.ocr

Augmenter that apply ocr error simulation to textual input.

class nlpaug.augmenter.char.ocr.**OcrAug**(*name='OCR_Aug'*, *aug_char_min=2*, *aug_char_max=10*, *aug_char_p=0.3*, *aug_word_p=0.3*, *aug_word_min=1*, *aug_word_max=10*, *stopwords=None*, *tokenizer=None*, *reverse_tokenizer=None*, *verbose=0*, *stopwords_regex=None*, *min_char=1*, *dict_of_path=None*)

Bases: nlpaug.augmenter.char.char_augmenter.CharAugmenter

Augmenter that simulate ocr error by random values. For example, OCR may recognize I as 1 incorrectly. Pre-defined OCR mapping is leveraged to replace character by possible OCR error.

Parameters

- **aug_char_p** (*float*) – Percentage of character (per token) will be augmented.
- **aug_char_min** (*int*) – Minimum number of character will be augmented.
- **aug_char_max** (*int*) – Maximum number of character will be augmented. If None is passed, number of augmentation is calculated via aup_char_p. If calculated result from aug_char_p is smaller than aug_char_max, will use calculated result from aup_char_p. Otherwise, using aug_max.
- **aug_word_p** (*float*) – Percentage of word will be augmented.
- **aug_word_min** (*int*) – Minimum number of word will be augmented.
- **aug_word_max** (*int*) – Maximum number of word will be augmented. If None is passed, number of augmentation is calculated via aup_word_p. If calculated result from aug_word_p is smaller than aug_word_max, will use calculated result from aug_word_p. Otherwise, using aug_max.
- **min_char** (*int*) – If word less than this value, do not draw word for augmentation
- **stopwords** (*list*) – List of words which will be skipped from augment operation.
- **stopwords_regex** (*str*) – Regular expression for matching words which will be skipped from augment operation.
- **tokenizer** (*func*) – Customize tokenization process
- **reverse_tokenizer** (*func*) – Customize reverse of tokenization process
- **dict_of_path** (*obj*) – Use pre-defined dictionary by default. Pass either file path of dict to use custom mapping.
- **name** (*str*) – Name of this augmenter

```
>>> import nlpaug.augmenter.char as nac
>>> aug = nac.OcrAug()
```

augment (*data*, *n=1*, *num_thread=1*)

Parameters

- **data** (*object/list*) – Data for augmentation. It can be list of data (e.g. list of string or numpy) or single element (e.g. string or numpy). Numpy format only supports audio or spectrogram data. For text data, only support string or list of string.
- **n** (*int*) – Default is 1. Number of unique augmented output. Will be force to 1 if input is list of data
- **num_thread** (*int*) – Number of thread for data augmentation. Use this option when you are using CPU and n is larger than 1

Returns Augmented data

```
>>> augmented_data = aug.augment(data)
```

3.2.3 nlpaug.augmenter.char.random

Augmenter that apply random character error to textual input.

class nlpaug.augmenter.char.random.**RandomCharAug** (*action='substitute'*, *name='RandomChar_Aug'*, *aug_char_min=1*, *aug_char_max=10*, *aug_char_p=0.3*, *aug_word_p=0.3*, *aug_word_min=1*, *aug_word_max=10*, *include_upper_case=True*, *include_lower_case=True*, *include_numeric=True*, *min_char=4*, *swap_mode='adjacent'*, *spec_char='!@#$%^&*()_+'*, *stopwords=None*, *tokenizer=None*, *reverse_tokenizer=None*, *verbose=0*, *stopwords_regex=None*, *candidates=None*)

Bases: nlpaug.augmenter.char.char_augmenter.CharAugmenter

Augmenter that generate character error by random values. For example, people may type i as o incorrectly.

Parameters

- **action** (*str*) – Possible values are 'insert', 'substitute', 'swap' and 'delete'. If value is 'insert', a new character will be injected to randomly. If value is 'substitute', a random character will be replaced original character randomly. If value is 'swap', adjacent characters within sample word will be swapped randomly. If value is 'delete', character will be removed randomly.
- **aug_char_p** (*float*) – Percentage of character (per token) will be augmented.
- **aug_char_min** (*int*) – Minimum number of character will be augmented.
- **aug_char_max** (*int*) – Maximum number of character will be augmented. If None is passed, number of augmentation is calculated via aup_char_p. If calculated result from

aug_char_p is smaller than aug_char_max, will use calculated result from aup_char_p. Otherwise, using aug_max.

- **aug_word_p** (*float*) – Percentage of word will be augmented.
- **aug_word_min** (*int*) – Minimum number of word will be augmented.
- **aug_word_max** (*int*) – Maximum number of word will be augmented. If None is passed, number of augmentation is calculated via aup_word_p. If calculated result from aug_word_p is smaller than aug_word_max, will use calculated result from aug_word_p. Otherwise, using aug_max.
- **include_upper_case** (*bool*) – If True, upper case character may be included in augmented data. If 'candidates' value is provided, this param will be ignored.
- **include_lower_case** (*bool*) – If True, lower case character may be included in augmented data. If 'candidates' value is provided, this param will be ignored.
- **include_numeric** (*bool*) – If True, numeric character may be included in augmented data. If 'candidates' value is provided, this param will be ignored.
- **min_char** (*int*) – If word less than this value, do not draw word for augmentation
- **swap_mode** – When action is 'swap', you may pass 'adjacent', 'middle' or 'random'. 'adjacent' means swap action only consider adjacent character (within same word). 'middle' means swap action consider adjacent character but not the first and last character of word. 'random' means swap action will be executed without constraint.
- **spec_char** (*str*) – Special character may be included in augmented data. If 'candidates' value is provided, this param will be ignored.
- **stopwords** (*list*) – List of words which will be skipped from augment operation.
- **stopwords_regex** (*str*) – Regular expression for matching words which will be skipped from augment operation.
- **tokenizer** (*func*) – Customize tokenization process
- **reverse_tokenizer** (*func*) – Customize reverse of tokenization process
- **candidates** (*List*) – List of string for augmentation. E.g. ['AAA', '11', '==='']. If values is provided, *include_upper_case*, *include_lower_case*, *include_numeric* and *spec_char* will be ignored.
- **name** (*str*) – Name of this augmenter.

```
>>> import nlpaug.augmenter.char as nac
>>> aug = nac.RandomCharAug()
```

augment(*data*, *n=1*, *num_thread=1*)

Parameters

- **data** (*object/list*) – Data for augmentation. It can be list of data (e.g. list of string or numpy) or single element (e.g. string or numpy). Numpy format only supports audio or spectrogram data. For text data, only support string or list of string.
- **n** (*int*) – Default is 1. Number of unique augmented output. Will be force to 1 if input is list of data
- **num_thread** (*int*) – Number of thread for data augmentation. Use this option when you are using CPU and n is larger than 1

Returns Augmented data

```
>>> augmented_data = aug.augment(data)
```

3.3 Sentence Augmenter

3.3.1 nlpaug.augmenter.sentence.abst_summ

Augmenter that apply operation (sentence level) to textual input based on abstractive summarization.

class nlpaug.augmenter.sentence.abst_summ.**AbstSummAug**(*model_path='t5-base'*, *tokenizer_path='t5-base'*, *min_length=20*, *max_length=50*, *batch_size=32*, *temperature=1.0*, *top_k=50*, *top_p=0.9*, *name='AbstSumm_Aug'*, *device='cpu'*, *force_reload=False*, *verbose=0*, *use_custom_api=True*)

Bases: nlpaug.augmenter.sentence.sentence_augmenter.SentenceAugmenter

Augmenter that leverage contextual word embeddings to find top n similar word for augmentation.

Parameters

- **model_path** (*str*) – Model name or model path. It used transformers to load the model. Tested 'facebook/bart-large-cnn', t5-small', 't5-base' and 't5-large'. For models, you can visit https://huggingface.co/models?filter=summarization
- **batch_size** (*int*) – Batch size.
- **min_length** (*int*) – The min length of output text.
- **max_length** (*int*) – The max length of output text.
- **temperature** (*float*) – The value used to module the next token probabilities.
- **top_k** (*int*) – The number of highest probability vocabulary tokens to keep for top-k-filtering.
- **top_p** (*float*) – If set to float < 1, only the most probable tokens with probabilities that add up to *top_p* or higher are kept for generation.
- **device** (*str*) – Default value is CPU. If value is CPU, it uses CPU for processing. If value is CUDA, it uses GPU for processing. Possible values include 'cuda' and 'cpu'. (May able to use other options)
- **force_reload** (*bool*) – Force reload the contextual word embeddings model to memory when initialize the class. Default value is False and suggesting to keep it as False if performance is the consideration.
- **name** (*str*) – Name of this augmenter

```
>>> import nlpaug.augmenter.sentence as nas
>>> aug = nas.AbstSummAug()
```

augment(*data*, *n=1*, *num_thread=1*)

Parameters

- **data** (*object/list*) – Data for augmentation. It can be list of data (e.g. list of string or numpy) or single element (e.g. string or numpy). Numpy format only supports audio or spectrogram data. For text data, only support string or list of string.
- **n** (*int*) – Default is 1. Number of unique augmented output. Will be force to 1 if input is list of data
- **num_thread** (*int*) – Number of thread for data augmentation. Use this option when you are using CPU and n is larger than 1

Returns Augmented data

```
>>> augmented_data = aug.augment(data)
```

3.3.2 nlpaug.augmenter.sentence.context_word_embs_sentence

Augmenter that apply operation (sentence level) to textual input based on contextual word embeddings.

class nlpaug.augmenter.sentence.context_word_embs_sentence.**ContextualWordEmbsForSentenceAu**

Bases: nlpaug.augmenter.sentence.sentence_augmenter.SentenceAugmenter

Augmenter that leverage contextual word embeddings to find top n similar word for augmentation.

Parameters

- **model_path** (*str*) – Model name or model path. It used transformers to load the model. Tested 'gpt2', 'distilgpt2'.
- **model_type** (*str*) – Type of model. For XLNet model, use 'xlnet'. For GPT2 or distilgpt2 model, use 'gpt'. If no value is provided, will determine from model name.
- **batch_size** (*int*) – Batch size.
- **min_length** (*int*) – The min length of output text.
- **max_length** (*int*) – The max length of output text.
- **temperature** (*float*) – The value used to module the next token probabilities.
- **top_k** (*int*) – The number of highest probability vocabulary tokens to keep for top-k-filtering.

- **top_p** (*float*) – If set to float < 1, only the most probable tokens with probabilities that add up to *top_p* or higher are kept for generation.
- **device** (*str*) – Default value is CPU. If value is CPU, it uses CPU for processing. If value is CUDA, it uses GPU for processing. Possible values include 'cuda' and 'cpu'. (May able to use other options)
- **force_reload** (*bool*) – Force reload the contextual word embeddings model to memory when initialize the class. Default value is False and suggesting to keep it as False if performance is the consideration.
- **silence** (*bool*) – Default is True. transformers library will print out warning message when leveraing pre-trained model. Set True to disable the expected warning message.
- **name** (*str*) – Name of this augmenter

```
>>> import nlpaug.augmenter.sentence as nas
>>> aug = nas.ContextualWordEmbsForSentenceAug()
```

augment(*data*, *n=1*, *num_thread=1*)

Parameters

- **data** (*object/list*) – Data for augmentation. It can be list of data (e.g. list of string or numpy) or single element (e.g. string or numpy). Numpy format only supports audio or spectrogram data. For text data, only support string or list of string.
- **n** (*int*) – Default is 1. Number of unique augmented output. Will be force to 1 if input is list of data
- **num_thread** (*int*) – Number of thread for data augmentation. Use this option when you are using CPU and n is larger than 1

Returns Augmented data

```
>>> augmented_data = aug.augment(data)
```

3.3.3 nlpaug.augmenter.sentence.lambada

Augmenter that apply operation (sentence level) to textual input based on abstractive summarization.

class nlpaug.augmenter.sentence.lambada.**LambadaAug**(*model_dir*, *threshold=None*, *min_length=100*, *max_length=300*, *batch_size=16*, *temperature=1.0*, *top_k=50*, *top_p=0.9*, *repetition_penalty=1.0*, *name='Lambada_Aug'*, *device='cpu'*, *force_reload=False*, *verbose=0*)

Bases: nlpaug.augmenter.sentence.sentence_augmenter.SentenceAugmenter

Augmenter that leverage contextual word embeddings to find top n similar word for augmentation.

Parameters

- **model_dir** (*str*) – Directory of model. It is generated from train_lambada.sh under scritps folders.n
- **threshold** (*float*) – The threshold of classification probabilty for accpeting generated text. Return all result if threshold is None.

- **batch_size** (*int*) – Batch size.
- **min_length** (*int*) – The min length of output text.
- **max_length** (*int*) – The max length of output text.
- **temperature** (*float*) – The value used to module the next token probabilities.
- **top_k** (*int*) – The number of highest probability vocabulary tokens to keep for top-k-filtering.
- **top_p** (*float*) – If set to float < 1, only the most probable tokens with probabilities that add up to *top_p* or higher are kept for generation.

:param float repetition_penalty : The parameter for repetition penalty. 1.0 means no penalty. :param str device: Default value is CPU. If value is CPU, it uses CPU for processing. If value is CUDA, it uses GPU

for processing. Possible values include 'cuda' and 'cpu'.

Parameters

- **force_reload** (*bool*) – Force reload the contextual word embeddings model to memory when initialize the class. Default value is False and suggesting to keep it as False if performance is the consideration.
- **name** (*str*) – Name of this augmenter

```
>>> import nlpaug.augmenter.sentence as nas
>>> aug = nas.LambadaAug()
```

augment (*data*, *n=1*, *num_thread=1*)

Parameters

- **data** (*object/list*) – Data for augmentation. It can be list of data (e.g. list of string or numpy) or single element (e.g. string or numpy). Numpy format only supports audio or spectrogram data. For text data, only support string or list of string.
- **n** (*int*) – Default is 1. Number of unique augmented output. Will be force to 1 if input is list of data
- **num_thread** (*int*) – Number of thread for data augmentation. Use this option when you are using CPU and n is larger than 1

Returns Augmented data

```
>>> augmented_data = aug.augment(data)
```

3.3.4 nlpaug.augmenter.sentence.random

Augmenter that apply operation (sentence level) to textual input based on abstractive summarization.

class nlpaug.augmenter.sentence.random.**RandomSentAug**(*mode='neighbor'*, *action='swap'*, *name='RandomSent_Aug'*, *aug_min=1*, *aug_max=10*, *aug_p=0.3*, *tokenizer=None*, *verbose=0*)

Bases: nlpaug.augmenter.sentence.sentence_augmenter.SentenceAugmenter

Augmenter that apply randomly behavior for augmentation.

Parameters

- **mode** (*str*) – Shuffle sentence to left, right, neighbor or random position. For *left*, target sentence will be swapped with left sentnece. For *right*, target sentence will be swapped with right sentnece. For *neighbor*, target sentence will be swapped with left or right sentnece radomly. For *random*, target sentence will be swapped with any sentnece randomly.
- **aug_p** (*float*) – Percentage of sentence will be augmented.
- **aug_min** (*int*) – Minimum number of sentence will be augmented.
- **aug_max** (*int*) – Maximum number of sentence will be augmented. If None is passed, number of augmentation is calculated via aup_p. If calculated result from aug_p is smaller than aug_max, will use calculated result from aug_p. Otherwise, using aug_max.
- **tokenizer** (*func*) – Customize tokenization process
- **name** (*str*) – Name of this augmenter

```
>>> import nlpaug.augmenter.sentence as nas
>>> aug = nas.RandomSentAug()
```

augment (*data*, *n=1*, *num_thread=1*)

Parameters

- **data** (*object/list*) – Data for augmentation. It can be list of data (e.g. list of string or numpy) or single element (e.g. string or numpy). Numpy format only supports audio or spectrogram data. For text data, only support string or list of string.
- **n** (*int*) – Default is 1. Number of unique augmented output. Will be force to 1 if input is list of data
- **num_thread** (*int*) – Number of thread for data augmentation. Use this option when you are using CPU and n is larger than 1

Returns Augmented data

```
>>> augmented_data = aug.augment(data)
```

3.4 Spectrogram Augmenter

3.4.1 nlpaug.augmenter.spectrogram.frequency_masking

class nlpaug.augmenter.spectrogram.frequency_masking.**FrequencyMaskingAug**(*name='FrequencyMaskin*
zone=(0.2, 0.8), coverage=1.0, factor=(40, 80), verbose=0, silence=False, stateless=True)

Bases: nlpaug.augmenter.spectrogram.spectrogram_augmenter.SpectrogramAugmenter

Augmenter that mask spectrogram based on frequency by random values.

Parameters

- **zone** (*tuple*) – Default value is (0.2, 0.8). Assign a zone for augmentation. By default, no any augmentation will be applied in first 20% and last 20% of whole audio.
- **coverage** (*float*) – Default value is 1 and value should be between 0 and 1. Portion of augmentation. If *1* is assigned, augment operation will be applied to target audio segment. For example, the audio duration is 60 seconds while zone and coverage are (0.2, 0.8) and 0.7 respectively. 42 seconds ((0.8-0.2)*0.7*60) audio will be augmented.
- **factor** (*tuple*) – Default value is (40, 80) and value should not exceed number of mel frequency channels. Factor value will be picked within the range of this tuple value. Mask range will be between [0, v - factor) while v is the number of mel frequency channels.
- **name** (*str*) – Name of this augmenter

```
>>> import nlpaug.augmenter.spectogram as nas
>>> aug = nas.FrequencyMaskingAug()
```

augment(*data*, *n=1*, *num_thread=1*)

Parameters

- **data** (*object/list*) – Data for augmentation. It can be list of data (e.g. list of string or numpy) or single element (e.g. string or numpy). Numpy format only supports audio or spectrogram data. For text data, only support string or list of string.
- **n** (*int*) – Default is 1. Number of unique augmented output. Will be force to 1 if input is list of data
- **num_thread** (*int*) – Number of thread for data augmentation. Use this option when you are using CPU and n is larger than 1

Returns Augmented data

```
>>> augmented_data = aug.augment(data)
```

substitute(*data*)

https://arxiv.org/pdf/1904.08779.pdf, https://arxiv.org/pdf/2001.01401.pdf Frequency masking is applied so that f consecutive mel frequency channels [f0, f0 + f) are masked, where f is first chosen from a uniform distribution from 0 to the frequency mask parameter F, and f0 is chosen from [0, v - f). v is the number of mel frequency channels.

3.4.2 nlpaug.augmenter.spectrogram.time_masking

class nlpaug.augmenter.spectrogram.time_masking.**TimeMaskingAug**(*name='TimeMasking_Aug'*, *zone=(0.2, 0.8)*, *coverage=1.0*, *verbose=0*, *silence=False*, *stateless=True*)

Bases: nlpaug.augmenter.spectrogram.spectrogram_augmenter.SpectrogramAugmenter

Augmenter that mask spectrogram based on frequency by random values.

Parameters

- **zone** (*tuple*) – Default value is (0.2, 0.8). Assign a zone for augmentation. By default, no any augmentation will be applied in first 20% and last 20% of whole audio.
- **coverage** (*float*) – Default value is 1 and value should be between 0 and 1. Portion of augmentation. If *1* is assigned, augment operation will be applied to target audio segment. For example, the audio duration is 60 seconds while zone and coverage are (0.2, 0.8) and 0.7 respectively. 42 seconds ((0.8-0.2)*0.7*60) audio will be chosen for augmentation.
- **name** (*str*) – Name of this augmenter

```
>>> import nlpaug.augmenter.spectogram as nas
>>> aug = nas.TimeMaskingAug()
```

augment(*data*, *n=1*, *num_thread=1*)

Parameters

- **data** (*object/list*) – Data for augmentation. It can be list of data (e.g. list of string or numpy) or single element (e.g. string or numpy). Numpy format only supports audio or spectrogram data. For text data, only support string or list of string.
- **n** (*int*) – Default is 1. Number of unique augmented output. Will be force to 1 if input is list of data
- **num_thread** (*int*) – Number of thread for data augmentation. Use this option when you are using CPU and n is larger than 1

Returns Augmented data

```
>>> augmented_data = aug.augment(data)
```

substitute(*data*)

From: https://arxiv.org/pdf/1904.08779.pdf, Time masking is applied so that t consecutive time steps [t0, t0 + t) are masked, where t is first chosen from a uniform distribution from 0 to the time mask parameter T, and t0 is chosen from [0, tau - t).

3.5 Word Augmenter

3.5.1 nlpaug.augmenter.word.antonym

Augmenter that apply semantic meaning based to textual input.

class nlpaug.augmenter.word.antonym.**AntonymAug**(*name='Antonym_Aug'*, *aug_min=1*, *aug_max=10*, *aug_p=0.3*, *lang='eng'*, *stopwords=None*, *tokenizer=None*, *reverse_tokenizer=None*, *stopwords_regex=None*, *verbose=0*)

Bases: nlpaug.augmenter.word.word_augmenter.WordAugmenter

Augmenter that leverage semantic meaning to substitute word.

Parameters

- **lang** (*str*) – Language of your text. Default value is 'eng'.
- **aug_p** (*float*) – Percentage of word will be augmented.
- **aug_min** (*int*) – Minimum number of word will be augmented.
- **aug_max** (*int*) – Maximum number of word will be augmented. If None is passed, number of augmentation is calculated via aup_p. If calculated result from aug_p is smaller than aug_max, will use calculated result from aug_p. Otherwise, using aug_max.
- **stopwords** (*list*) – List of words which will be skipped from augment operation.
- **stopwords_regex** (*str*) – Regular expression for matching words which will be skipped from augment operation.
- **tokenizer** (*func*) – Customize tokenization process
- **reverse_tokenizer** (*func*) – Customize reverse of tokenization process
- **name** (*str*) – Name of this augmenter

```
>>> import nlpaug.augmenter.word as naw
>>> aug = naw.AntonymAug()
```

augment(*data*, *n=1*, *num_thread=1*)

Parameters

- **data** (*object/list*) – Data for augmentation. It can be list of data (e.g. list of string or numpy) or single element (e.g. string or numpy). Numpy format only supports audio or spectrogram data. For text data, only support string or list of string.
- **n** (*int*) – Default is 1. Number of unique augmented output. Will be force to 1 if input is list of data
- **num_thread** (*int*) – Number of thread for data augmentation. Use this option when you are using CPU and n is larger than 1

Returns Augmented data

```
>>> augmented_data = aug.augment(data)
```

3.5.2 nlpaug.augmenter.word.back_translation

Augmenter that apply operation (word level) to textual input based on back translation.

class nlpaug.augmenter.word.back_translation.**BackTranslationAug**(*from_model_name='facebook/wmt19-en-de'*, *to_model_name='facebook/wmt19-de-en'*, *name='BackTranslationAug'*, *device='cpu'*, *batch_size=32*, *max_length=300*, *force_reload=False*, *verbose=0*)

Bases: nlpaug.augmenter.word.word_augmenter.WordAugmenter

Augmenter that leverage two translation models for augmentation. For example, the source is English. This augmenter translate source to German and translating it back to English. For detail, you may visit https://towardsdatascience.com/data-augmentation-in-nlp-2801a34dfc28

Parameters

- **from_model_name** (*str*) – Any model from https://huggingface.co/models?filter=translation&search=Helsinki-NLP. As long as from_model_name is pair with to_model_name. For example, from_model_name is English to Japanese, then to_model_name should be Japanese to English.
- **to_model_name** (*str*) – Any model from https://huggingface.co/models?filter=translation&search=Helsinki-NLP.
- **device** (*str*) – Default value is CPU. If value is CPU, it uses CPU for processing. If value is CUDA, it uses GPU for processing. Possible values include 'cuda' and 'cpu'. (May able to use other options)
- **force_reload** (*bool*) – Force reload the contextual word embeddings model to memory when initialize the class. Default value is False and suggesting to keep it as False if performance is the consideration.
- **batch_size** (*int*) – Batch size.
- **max_length** (*int*) – The max length of output text.
- **name** (*str*) – Name of this augmenter

```
>>> import nlpaug.augmenter.word as naw
>>> aug = naw.BackTranslationAug()
```

augment(*data*, *n=1*, *num_thread=1*)

Parameters

- **data** (*object/list*) – Data for augmentation. It can be list of data (e.g. list of string or numpy) or single element (e.g. string or numpy). Numpy format only supports audio or spectrogram data. For text data, only support string or list of string.
- **n** (*int*) – Default is 1. Number of unique augmented output. Will be force to 1 if input is list of data
- **num_thread** (*int*) – Number of thread for data augmentation. Use this option when you are using CPU and n is larger than 1

Returns Augmented data

```
>>> augmented_data = aug.augment(data)
```

3.5.3 nlpaug.augmenter.word.context_word_embs

Augmenter that apply operation (word level) to textual input based on contextual word embeddings.

class nlpaug.augmenter.word.context_word_embs.**ContextualWordEmbsAug**(*model_path='bert-base-uncased'*, *model_type=''*, *action='substitute'*, *top_k=100*, *name='ContextualWordEmbs_Au*
aug_min=1, *aug_max=10*, *aug_p=0.3*, *stopwords=None*, *batch_size=32*, *device='cpu'*, *force_reload=False*, *stopwords_regex=None*, *verbose=0*, *silence=True*, *use_custom_api=True*)

Bases: nlpaug.augmenter.word.word_augmenter.WordAugmenter

Augmenter that leverage contextual word embeddings to find top n similar word for augmentation.

Parameters

- **model_path** (*str*) – Model name or model path. It used transformers to load the model. Tested 'bert-base-uncased', 'bert-base-cased', 'distilbert-base-uncased', 'roberta-base', 'distilroberta-base', 'facebook/bart-base', 'squeezebert/squeezebert-uncased'.
- **model_type** (*str*) – Type of model. For BERT model, use 'bert'. For RoBERTa/LongFormer model, use 'roberta'. For BART model, use 'bart'. If no value is provided, will determine from model name.
- **action** (*str*) – Either 'insert or 'substitute'. If value is 'insert', a new word will be injected to random position according to contextual word embeddings calculation. If value is 'substitute', word will be replaced according to contextual embeddings calculation
- **top_k** (*int*) – Controlling lucky draw pool. Top k score token will be used for augmentation. Larger k, more token can be used. Default value is 100. If value is None which means using all possible tokens.
- **aug_p** (*float*) – Percentage of word will be augmented.
- **aug_min** (*int*) – Minimum number of word will be augmented.

- **aug_max** (*int*) – Maximum number of word will be augmented. If None is passed, number of augmentation is calculated via aup_p. If calculated result from aug_p is smaller than aug_max, will use calculated result from aug_p. Otherwise, using aug_max.
- **stopwords** (*list*) – List of words which will be skipped from augment operation. Do NOT include the UNKNOWN word. UNKNOWN word of BERT is [UNK]. UNKNOWN word of RoBERTa and BART is <unk>.
- **stopwords_regex** (*str*) – Regular expression for matching words which will be skipped from augment operation.
- **device** (*str*) – Default value is CPU. If value is CPU, it uses CPU for processing. If value is CUDA, it uses GPU for processing. Possible values include 'cuda' and 'cpu'. (May able to use other options)
- **batch_size** (*int*) – Batch size.
- **force_reload** (*bool*) – Force reload the contextual word embeddings model to memory when initialize the class. Default value is False and suggesting to keep it as False if performance is the consideration.
- **silence** (*bool*) – Default is True. transformers library will print out warning message when leveraing pre-trained model. Set True to disable the expected warning message.
- **name** (*str*) – Name of this augmenter

```
>>> import nlpaug.augmenter.word as naw
>>> aug = naw.ContextualWordEmbsAug()
```

augment (*data*, *n=1*, *num_thread=1*)

Parameters

- **data** (*object/list*) – Data for augmentation. It can be list of data (e.g. list of string or numpy) or single element (e.g. string or numpy). Numpy format only supports audio or spectrogram data. For text data, only support string or list of string.
- **n** (*int*) – Default is 1. Number of unique augmented output. Will be force to 1 if input is list of data
- **num_thread** (*int*) – Number of thread for data augmentation. Use this option when you are using CPU and n is larger than 1

Returns Augmented data

```
>>> augmented_data = aug.augment(data)
```

device = None

TODO: Reserve 2 spaces (e.g. [CLS], [SEP]) is not enough as it hit CUDA error in batch processing mode. Therefore, forcing to reserve 5 times of reserved spaces (i.e. 5)

3.5.4 nlpaug.augmenter.word.random

Augmenter that apply random word operation to textual input.

class nlpaug.augmenter.word.random.**RandomWordAug**(*action='delete'*, *name='RandomWord_Aug'*, *aug_min=1*, *aug_max=10*, *aug_p=0.3*, *stopwords=None*, *target_words=None*, *tokenizer=None*, *reverse_tokenizer=None*, *stopwords_regex=None*, *verbose=0*)

Bases: nlpaug.augmenter.word.word_augmenter.WordAugmenter

Augmenter that apply randomly behavior for augmentation.

Parameters

- **action** (*str*) – 'substitute', 'swap', 'delete' or 'crop'. If value is 'swap', adjacent words will be swapped randomly. If value is 'delete', word will be removed randomly. If value is 'crop', a set of contunous word will be removed randomly.
- **aug_p** (*float*) – Percentage of word will be augmented.
- **aug_min** (*int*) – Minimum number of word will be augmented.
- **aug_max** (*int*) – Maximum number of word will be augmented. If None is passed, number of augmentation is calculated via aup_p. If calculated result from aug_p is smaller than aug_max, will use calculated result from aug_p. Otherwise, using aug_max.
- **stopwords** (*list*) – List of words which will be skipped from augment operation. Not effective if action is 'crop'
- **stopwords_regex** (*str*) – Regular expression for matching words which will be skipped from augment operation. Not effective if action is 'crop'
- **target_words** (*list*) – List of word for replacement (used for substitute operation only). Default value is _.
- **tokenizer** (*func*) – Customize tokenization process
- **reverse_tokenizer** (*func*) – Customize reverse of tokenization process
- **name** (*str*) – Name of this augmenter

```
>>> import nlpaug.augmenter.word as naw
>>> aug = naw.RandomWordAug()
```

augment(*data*, *n=1*, *num_thread=1*)

Parameters

- **data** (*object/list*) – Data for augmentation. It can be list of data (e.g. list of string or numpy) or single element (e.g. string or numpy). Numpy format only supports audio or spectrogram data. For text data, only support string or list of string.
- **n** (*int*) – Default is 1. Number of unique augmented output. Will be force to 1 if input is list of data
- **num_thread** (*int*) – Number of thread for data augmentation. Use this option when you are using CPU and n is larger than 1

Returns Augmented data

```
>>> augmented_data = aug.augment(data)
```

3.5.5 nlpaug.augmenter.word.reserved

Augmenter that apply target word replacement operation to textual input.

class nlpaug.augmenter.word.reserved.**ReservedAug**(*reserved_tokens*, *action='substitute'*, *case_sensitive=True*, *name='Reserved_Aug'*, *aug_min=1*, *aug_max=10*, *aug_p=0.3*, *tokenizer=None*, *reverse_tokenizer=None*, *verbose=0*, *generate_all_combinations=False*)

Bases: nlpaug.augmenter.word.word_augmenter.WordAugmenter

Augmenter that apply target word replacement for augmentation. Can also be used to generate all possible combinations. :param float aug_p: Percentage of word will be augmented. :param int aug_min: Minimum number of word will be augmented. :param int aug_max: Maximum number of word will be augmented. If None is passed, number of augmentation is

calculated via aup_p. If calculated result from aug_p is smaller than aug_max, will use calculated result from aug_p. Otherwise, using aug_max.

Parameters

- **reserved_tokens** (*list*) – A list of swappable tokens (a list of list). For example, "FWD", "Fwd" and "FW" are referring to "foward" in email communcation while "Sincerely" and "Best Regards" treated as same meaning. The input should be [["FWD", "Fwd", "FW"], ["Sincerely", "Best Regards"]].
- **case_sensitive** (*bool*) – Default is True. If True, it will only replace alternative token if all cases are same.
- **generate_all_combinations** (*bool*) – Default is False. If True, all the possible combinations of sentences possible with reserved_tokens will be returned.
- **tokenizer** (*func*) – Customize tokenization process
- **reverse_tokenizer** (*func*) – Customize reverse of tokenization process
- **name** (*str*) – Name of this augmenter

```
>>> import nlpaug.augmenter.word as naw
>>> aug = naw.ReservedAug()
```

augment(*data*, *n=1*, *num_thread=1*)

Parameters

- **data** (*object/list*) – Data for augmentation. It can be list of data (e.g. list of string or numpy) or single element (e.g. string or numpy). Numpy format only supports audio or spectrogram data. For text data, only support string or list of string.
- **n** (*int*) – Default is 1. Number of unique augmented output. Will be force to 1 if input is list of data
- **num_thread** (*int*) – Number of thread for data augmentation. Use this option when you are using CPU and n is larger than 1

Returns Augmented data

```
>>> augmented_data = aug.augment(data)
```

3.5.6 nlpaug.augmenter.word.spelling

Augmenter that apply spelling error simulation to textual input.

class nlpaug.augmenter.word.spelling.**SpellingAug**(*dict_path=None*, *name='Spelling_Aug'*, *aug_min=1*, *aug_max=10*, *aug_p=0.3*, *stopwords=None*, *tokenizer=None*, *reverse_tokenizer=None*, *include_reverse=True*, *stopwords_regex=None*, *verbose=0*)

Bases: nlpaug.augmenter.word.word_augmenter.WordAugmenter

Augmenter that leverage pre-defined spelling mistake dictionary to simulate spelling mistake.

Parameters

- **dict_path** (*str*) – Path of misspelling dictionary
- **aug_p** (*float*) – Percentage of word will be augmented.
- **aug_min** (*int*) – Minimum number of word will be augmented.
- **aug_max** (*int*) – Maximum number of word will be augmented. If None is passed, number of augmentation is calculated via aup_p. If calculated result from aug_p is smaller than aug_max, will use calculated result from aug_p. Otherwise, using aug_max.
- **stopwords** (*list*) – List of words which will be skipped from augment operation.
- **stopwords_regex** (*str*) – Regular expression for matching words which will be skipped from augment operation.
- **tokenizer** (*func*) – Customize tokenization process
- **reverse_tokenizer** (*func*) – Customize reverse of tokenization process
- **name** (*str*) – Name of this augmenter

```
>>> import nlpaug.augmenter.word as naw
>>> aug = naw.SpellingAug(dict_path='./spelling_en.txt')
```

augment(*data*, *n=1*, *num_thread=1*)

Parameters

- **data** (*object/list*) – Data for augmentation. It can be list of data (e.g. list of string or numpy) or single element (e.g. string or numpy). Numpy format only supports audio or spectrogram data. For text data, only support string or list of string.
- **n** (*int*) – Default is 1. Number of unique augmented output. Will be force to 1 if input is list of data
- **num_thread** (*int*) – Number of thread for data augmentation. Use this option when you are using CPU and n is larger than 1

Returns Augmented data

```
>>> augmented_data = aug.augment(data)
```

3.5.7 nlpaug.augmenter.word.split

Augmenter that apply word splitting operation to textual input.

class nlpaug.augmenter.word.split.**SplitAug**(*name='Split_Aug'*, *aug_min=1*, *aug_max=10*, *aug_p=0.3*, *min_char=4*, *stopwords=None*, *tokenizer=None*, *reverse_tokenizer=None*, *stopwords_regex=None*, *verbose=0*)

Bases: nlpaug.augmenter.word.word_augmenter.WordAugmenter

Augmenter that apply word splitting for augmentation.

Parameters

- **aug_p** (*float*) – Percentage of word will be augmented.
- **aug_min** (*int*) – Minimum number of word will be augmented.
- **aug_max** (*int*) – Maximum number of word will be augmented. If None is passed, number of augmentation is calculated via aup_p. If calculated result from aug_p is smaller than aug_max, will use calculated result from aug_p. Otherwise, using aug_max.
- **min_char** (*int*) – If word less than this value, do not draw word for augmentation
- **stopwords** (*list*) – List of words which will be skipped from augment operation.
- **stopwords_regex** (*str*) – Regular expression for matching words which will be skipped from augment operation.
- **tokenizer** (*func*) – Customize tokenization process
- **reverse_tokenizer** (*func*) – Customize reverse of tokenization process
- **name** (*str*) – Name of this augmenter

```
>>> import nlpaug.augmenter.word as naw
>>> aug = naw.SplitAug()
```

augment(*data*, *n=1*, *num_thread=1*)

Parameters

- **data** (*object/list*) – Data for augmentation. It can be list of data (e.g. list of string or numpy) or single element (e.g. string or numpy). Numpy format only supports audio or spectrogram data. For text data, only support string or list of string.
- **n** (*int*) – Default is 1. Number of unique augmented output. Will be force to 1 if input is list of data
- **num_thread** (*int*) – Number of thread for data augmentation. Use this option when you are using CPU and n is larger than 1

Returns Augmented data

```
>>> augmented_data = aug.augment(data)
```

3.5.8 nlpaug.augmenter.word.synonym

Augmenter that apply semantic meaning based to textual input.

class nlpaug.augmenter.word.synonym.**SynonymAug**(*aug_src='wordnet'*, *model_path=None*, *name='Synonym_Aug'*, *aug_min=1*, *aug_max=10*, *aug_p=0.3*, *lang='eng'*, *stopwords=None*, *tokenizer=None*, *reverse_tokenizer=None*, *stopwords_regex=None*, *force_reload=False*, *verbose=0*)

Bases: nlpaug.augmenter.word.word_augmenter.WordAugmenter

Augmenter that leverage semantic meaning to substitute word.

Parameters

- **aug_src** (*str*) – Support 'wordnet' and 'ppdb' .
- **model_path** (*str*) – Path of dictionary. Mandatory field if using PPDB as data source
- **lang** (*str*) – Language of your text. Default value is 'eng'. For *wordnet*, you can choose lang from this list http://compling.hss.ntu.edu.sg/omw/. For *ppdb*, you simply download corresponding langauge pack from http://paraphrase.org/#/download.
- **aug_p** (*float*) – Percentage of word will be augmented.
- **aug_min** (*int*) – Minimum number of word will be augmented.
- **aug_max** (*int*) – Maximum number of word will be augmented. If None is passed, number of augmentation is calculated via aup_p. If calculated result from aug_p is smaller than aug_max, will use calculated result from aug_p. Otherwise, using aug_max.
- **stopwords** (*list*) – List of words which will be skipped from augment operation.
- **stopwords_regex** (*str*) – Regular expression for matching words which will be skipped from augment operation.
- **tokenizer** (*func*) – Customize tokenization process
- **reverse_tokenizer** (*func*) – Customize reverse of tokenization process
- **force_reload** (*bool*) – Force reload model to memory when initialize the class. Default value is False and suggesting to keep it as False if performance is the consideration.
- **name** (*str*) – Name of this augmenter

```
>>> import nlpaug.augmenter.word as naw
>>> aug = naw.SynonymAug()
```

augment(*data*, *n=1*, *num_thread=1*)

Parameters

- **data** (*object/list*) – Data for augmentation. It can be list of data (e.g. list of string or numpy) or single element (e.g. string or numpy). Numpy format only supports audio or spectrogram data. For text data, only support string or list of string.
- **n** (*int*) – Default is 1. Number of unique augmented output. Will be force to 1 if input is list of data
- **num_thread** (*int*) – Number of thread for data augmentation. Use this option when you are using CPU and n is larger than 1

Returns Augmented data

```
>>> augmented_data = aug.augment(data)
```

3.5.9 nlpaug.augmenter.word.tfidf

Augmenter that apply TF-IDF based to textual input.

class nlpaug.augmenter.word.tfidf.**TfIdfAug**(*model_path='.'*, *action='substitute'*, *name='TfIdf_Aug'*, *aug_min=1*, *aug_max=10*, *aug_p=0.3*, *top_k=5*, *stopwords=None*, *tokenizer=None*, *reverse_tokenizer=None*, *stopwords_regex=None*, *verbose=0*)

Bases: nlpaug.augmenter.word.word_augmenter.WordAugmenter

Augmenter that leverage TF-IDF statistics to insert or substitute word.

Parameters

- **model_path** (*str*) – Downloaded model directory. Either model_path or model is must be provided
- **action** (*str*) – Either 'insert or 'substitute'. If value is 'insert', a new word will be injected to random position according to TF-IDF calculation. If value is 'substitute', word will be replaced according to TF-IDF calculation
- **top_k** (*int*) – Controlling lucky draw pool. Top k score token will be used for augmentation. Larger k, more token can be used. Default value is 5. If value is None which means using all possible tokens.
- **aug_p** (*float*) – Percentage of word will be augmented.
- **aug_min** (*int*) – Minimum number of word will be augmented.
- **aug_max** (*int*) – Maximum number of word will be augmented. If None is passed, number of augmentation is calculated via aup_p. If calculated result from aug_p is smaller than aug_max, will use calculated result from aug_p. Otherwise, using aug_max.
- **stopwords** (*list*) – List of words which will be skipped from augment operation.
- **stopwords_regex** (*str*) – Regular expression for matching words which will be skipped from augment operation.
- **tokenizer** (*func*) – Customize tokenization process
- **reverse_tokenizer** (*func*) – Customize reverse of tokenization process
- **name** (*str*) – Name of this augmenter

```
>>> import nlpaug.augmenter.word as naw
>>> aug = naw.TfIdfAug(model_path='.')
```

augment(*data*, *n=1*, *num_thread=1*)

Parameters

- **data** (*object/list*) – Data for augmentation. It can be list of data (e.g. list of string or numpy) or single element (e.g. string or numpy). Numpy format only supports audio or spectrogram data. For text data, only support string or list of string.
- **n** (*int*) – Default is 1. Number of unique augmented output. Will be force to 1 if input is list of data
- **num_thread** (*int*) – Number of thread for data augmentation. Use this option when you are using CPU and n is larger than 1

Returns Augmented data

```
>>> augmented_data = aug.augment(data)
```

3.5.10 nlpaug.augmenter.word.word_embs

Augmenter that apply operation to textual input based on word embeddings.

class nlpaug.augmenter.word.word_embs.**WordEmbsAug**(*model_type*, *model_path='.'*, *model=None*, *action='substitute'*, *name='WordEmbs_Aug'*, *aug_min=1*, *aug_max=10*, *aug_p=0.3*, *top_k=100*, *n_gram_separator='_'*, *stopwords=None*, *tokenizer=None*, *reverse_tokenizer=None*, *force_reload=False*, *stopwords_regex=None*, *verbose=0*, *skip_check=False*)

Bases: nlpaug.augmenter.word.word_augmenter.WordAugmenter

Augmenter that leverage word embeddings to find top n similar word for augmentation.

Parameters

- **model_type** (*str*) – Model type of word embeddings. Expected values include 'word2vec', 'glove' and 'fasttext'.
- **model_path** (*str*) – Downloaded model directory. Either model_path or model is must be provided
- **model** (*obj*) – Pre-loaded model (e.g. model class is nlpaug.model.word_embs.nmw.Word2vec(), nlpaug.model.word_embs.nmw.Glove() or nlpaug.model.word_embs.nmw.Fasttext())
- **action** (*str*) – Either 'insert or 'substitute'. If value is 'insert', a new word will be injected to random position according to word embeddings calculation. If value is 'substitute', word will be replaced according to word embeddings calculation
- **top_k** (*int*) – Controlling lucky draw pool. Top k score token will be used for augmentation. Larger k, more token can be used. Default value is 100. If value is None which means using all possible tokens. This attribute will be ignored when using "insert" action.
- **aug_p** (*float*) – Percentage of word will be augmented.
- **aug_min** (*int*) – Minimum number of word will be augmented.
- **aug_max** (*int*) – Maximum number of word will be augmented. If None is passed, number of augmentation is calculated via aup_p. If calculated result from aug_p is smaller than aug_max, will use calculated result from aug_p. Otherwise, using aug_max.
- **stopwords** (*list*) – List of words which will be skipped from augment operation.
- **stopwords_regex** (*str*) – Regular expression for matching words which will be skipped from augment operation.
- **tokenizer** (*func*) – Customize tokenization process
- **reverse_tokenizer** (*func*) – Customize reverse of tokenization process
- **force_reload** (*bool*) – If True, model will be loaded every time while it takes longer time for initialization.

- **skip_check** (*bool*) – Default is False. If True, no validation for size of vocabulary embedding.
- **name** (*str*) – Name of this augmenter

```
>>> import nlpaug.augmenter.word as naw
>>> aug = naw.WordEmbsAug(model_type='word2vec', model_path='.')
```

augment (*data*, *n=1*, *num_thread=1*)

Parameters

- **data** (*object/list*) – Data for augmentation. It can be list of data (e.g. list of string or numpy) or single element (e.g. string or numpy). Numpy format only supports audio or spectrogram data. For text data, only support string or list of string.
- **n** (*int*) – Default is 1. Number of unique augmented output. Will be force to 1 if input is list of data
- **num_thread** (*int*) – Number of thread for data augmentation. Use this option when you are using CPU and n is larger than 1

Returns Augmented data

```
>>> augmented_data = aug.augment(data)
```

CHAPTER 4

Flow

4.1 nlpaug.flow.sequential

Flow that apply augmentation sequentially.

class nlpaug.flow.sequential.**Sequential** (*flow=None*, *name='Sequential_Pipeline'*, *verbose=0*)

Bases: nlpaug.flow.pipeline.Pipeline

Flow that apply augmenters sequentially.

Parameters

- **flow** (*list*) – list of flow or augmenter
- **name** (*str*) – Name of this augmenter

```
>>> import nlpaug.flow as naf
>>> import nlpaug.augmenter.char as nac
>>> import nlpaug.augmenter.word as naw
>>> flow = naf.Sequential([nac.RandomCharAug(), naw.RandomWordAug()])
```

4.2 nlpaug.flow.sometimes

Flow that apply augmentation randomly.

class nlpaug.flow.sometimes.**Sometimes** (*flow=None*, *name='Sometimes_Pipeline'*, *aug_p=0.8*, *verbose=0*)

Bases: nlpaug.flow.pipeline.Pipeline

Flow that apply augmenters randomly.

Parameters

- **flow** (*list*) – list of flow or augmenter

- **aug_p** (*float*) – Percentage of pipeline will be executed.
- **name** (*str*) – Name of this augmenter

```
>>> import nlpaug.flow as naf
>>> import nlpaug.augmenter.char as nac
>>> import nlpaug.augmenter.word as naw
>>> flow = naf.Sometimes([nac.RandomCharAug(), naw.RandomWordAug()])
```

CHAPTER 5

Util

5.1 nlpaug.util.file.download

class nlpaug.util.file.download.**DownloadUtil**

Bases: object

Helper function for downloading external dependency

```
>>> from nlpaug.util.file.download import DownloadUtil
```

static download_fasttext(*model_name*, *dest_dir*)

Parameters

- **model_name** (*str*) – GloVe pre-trained model name. Possible values are 'wiki-news-300d-1M', 'wiki-news-300d-1M-subword', 'crawl-300d-2M' and 'crawl-300d-2M-subword'
- **dest_dir** (*str*) – Directory of saving file

```
>>> DownloadUtil.download_fasttext('glove.6B', '.')
```

static download_glove(*model_name*, *dest_dir*)

Parameters

- **model_name** (*str*) – GloVe pre-trained model name. Possible values are 'glove.6B', 'glove.42B.300d', 'glove.840B.300d' and 'glove.twitter.27B'
- **dest_dir** (*str*) – Directory of saving file

```
>>> DownloadUtil.download_glove('glove.6B', '.')
```

static download_word2vec(*dest_dir: str = '.'*)

Parameters **dest_dir** (*str*) – Directory of saving file

Returns Word2Vec C binary file named 'GoogleNews-vectors-negative300.bin'

```
>>> DownloadUtil.download_word2vec('.')
```

static unzip(*file_path*, *dest_dir=None*)

Parameters **file_path** (*str*) – File path for unzip

```
>>> DownloadUtil.unzip('zip_file.zip')
```

See modindex for API.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

n

Index

T

U

V

W