TimeseriesGenerator

```
keras.preprocessing.sequence.TimeseriesGenerator(data, targets, length, sampling_rate=1, stride=1
```

時系列データのデータのバッチを生成するためのユーティリティクラス.

このクラスは訓練や評価のためのバッチを生成するために，ストライドや履歴の長さ等のような時系列データとともに，等間隔に集められたデータ点のシーケンスを取り込みます.

引数

- **data**: 連続的なデータ点（タイムステップ）を含んだリストやNumpy配列のようなインデックス可能なジェネレータ．このデータは2次元である必要があり，軸0は時間の次元である事が期待されます.
- **targets**: データの中でタイムステップに対応するターゲット．データと同じ長さである必要があります.
- **length**: （タイムステップ数において）出力シーケンスの長さ.
- **sampling_rate**: シーケンス内で連続した独立の期間．レート `r` によって決まるタイムステップ `data[i]`, `data[i-r]`, ... `data[i - length]` はサンプルのシーケンス生成に使われます.
- **stride**: 連続した出力シーケンスの範囲．連続した出力サンプルはストライド `s` の値によって決まる `data[i]`, `data[i+s]`, `data[i+2*s]` などから出来ています.
- **start_index, end_index**: `start_index` より前または `end_index` より後のデータ点は出力シーケンスでは使われません．これはテストや検証のためにデータの一部を予約するのに便利です.
- **shuffle**: 出力サンプルをシャッフルするか，時系列順にするか
- **reverse**: 真理値： `true` なら各出力サンプルにおけるタイムステップが逆順になります.
- **batch_size**: 各バッチにおける時系列サンプル数（おそらく最後の1つを除きます）.

戻り値

Sequenceインスタンス.

例

```
from keras.preprocessing.sequence import TimeseriesGenerator
import numpy as np

data = np.array([[i] for i in range(50)])
targets = np.array([[i] for i in range(50)])

data_gen = TimeseriesGenerator(data, targets,
                               length=10, sampling_rate=2,
                               batch_size=2)
assert len(data_gen) == 20

batch_0 = data_gen[0]
x, y = batch_0
assert np.array_equal(x,
                      np.array([[[0], [2], [4], [6], [8]],
                                [[1], [3], [5], [7], [9]]]))
assert np.array_equal(y,
                      np.array([[10], [11]]))
```

---

pad_sequences

```
pad_sequences(sequences, maxlen=None, dtype='int32', padding='pre', truncating='pre', value=0.0)
```

シーケンスを同じ長さになるように詰めます.

`num_samples` シーケンスから構成されるリスト（スカラのリスト）をshape が `(num_samples, num_timesteps)` の2次元のNumpy 配列に変換します. `num_timesteps` は `maxlen` 引数が与えられれば `maxlen` に，与えられなければ最大のシーケンス長になります.

`num_timesteps` より短いシーケンスは，`value` でパディングされます.

`num_timesteps` より長いシーケンスは，指定された長さに切り詰められます. パディングと切り詰めの位置はそれぞれ `padding` と `truncating` によって決められます.

pre-paddingがデフォルトです.

引数

- **sequences**: リストのリスト，各要素はそれぞれシーケンスです.
- **maxlen**: 整数，シーケンスの最大長.
- **dtype**: 出力シーケンスの型.
- **padding**: 文字列，'pre'または'post'. 各シーケンスの前後どちらを埋めるか.
- **truncating**: 文字列，'pre'または'post'. `maxlen` より長いシーケンスの前後どちらを切り詰めるか.
- **value**: 浮動小数点数. パディングする値.

戻り値

- **x**: shapeが `(len(sequences), maxlen)` のNumpy配列.

Raises

- ValueError: `truncating` や `padding` が無効な値の場合，または `sequences` のエントリが無効なshapeの場合．

---

skipgrams

```
skipgrams(sequence, vocabulary_size, window_size=4, negative_samples=1.0, shuffle=True, categorica
```

skipgramの単語ペアを生成します．

この関数は単語インデックスのシーケンス（整数のリスト）を以下の形式の単語のタプルに変換します:

- （単語, 同じ文脈で出現する単語）, 1のラベル （正例）．
- （単語, 語彙中のランダムな単語）, 0のラベル （負例）．

Skipgramの詳細はMikolovらの論文を参照してください: Efficient Estimation of Word Representations in Vector Space

引数

- **sequence**: 単語のシーケンス（文）で，単語インデックス（整数）のリストとしてエンコードされたもの．`sampling_table` を使う場合，単語インデックスは参照するデータセットの中で単語のランクにあったランクである事が期待されます（例えば10は10番目に狂喜するトークンにエンコードされます）．インデックス0は無意味な語を期待され，スキップされます．
- **vocabulary_size**: 整数．可能な単語インデックスの最大値+1．
- **window_size**: 整数．サンプリングするウィンドウのサイズ（技術的には半分のウィンドウ）．単語 `w_i` のウィンドウは `[i - window_size, i + window_size+1]` になります．
- **negative_samples**: 0以上の浮動小数点数．0はネガティブサンプル数が0になります．1はネガティブサンプル数がポジティブサンプルと同じ数になります．
- **shuffle**: 単語の組を変える前にシャッフルするかどうか．
- **categorical**: 真理値．Falseならラベルは整数（例えば `[0, 1, 1 .. ]`）になり，`True` ならカテゴリカル，例えば `[[1,0],[0,1],[0,1] .. ]` になります．
- **sampling_table**: サイズが `vocabulary_size` の1次元配列．エントリiはインデックスiを持つ単語（データセット中でi番目に頻出する単語を想定します）のサンプリング確率です．
- **seed**: ランダムシード．

戻り値

couples, labels: `couples` は整数のペア，`labels` は0か1のいずれかです．

注意

慣例により，語彙の中でインデックスが0のものは単語ではなく，スキップされます．

---

make_sampling_table

```
make_sampling_table(size, sampling_factor=1e-05)
```

ワードランクベースの確率的なサンプリングテーブルを生成します.

`skipgrams` の `sampling_table` 引数を生成するために利用します. `sampling_table[i]` はデータセット中でi番目に頻出する単語をサンプリングする確率です（バランスを保つために，頻出語はこれより低い頻度でサンプリングされます）.

サンプリングの確率はword2vecで使われるサンプリング分布に従って生成されます：

```
p(word) = min(1, sqrt(word_frequency / sampling_factor) / (word_frequency / sampling_factor))
```

頻度（順位）の数値近似を得る（s=1の）ジップの法則に単語の頻度も従っていると仮定しています.

`frequency(rank) ~ 1/(rank * (log(rank) + gamma) + 1/2 - 1/(12*rank))` の `gamma` はオイラー・マスケローニ定数です.

引数 - **size**: 整数，サンプリング可能な語彙数. - **sampling_factor**: word2vecの式におけるサンプリング因子.

戻り値

長さが `size` の1次元のNumpy配列で，i番目の要素はランクiのワードがサンプリングされる確率です.