Обновляемый progressbar для программ на Python

Мод… …быстрого и расширяемого внедрения индикаторов выполнения (progressbar) во внешние интерфейсы про… …яя конечным пользователям визуальную индикацию хода вычислений или передачи данных. Он так… …тладки, как в качестве инструмента профилирования, так и в качестве способа отображения инф… …й задачи. Благодаря простоте использования библиотека также является идеальным кандидатом для… …ые курсы Python.

Содержание:

Проблемы, которые решает модуль tqdm.

Распространенной проблемой при программировании является наличие итерационных операций, когда мониторинг процесса желателен или даже выгоден. Распространенной стратегией мониторинга итераций, является включение дополнительного кода в цикл for для вывода текущего номера итерации или каких то дополнительных промежуточных данных. Тем не менее, есть много улучшений, которые можно было бы сделать в таком сценарии:

- отображение скорости итерации;
- отображение прошедшего и предполагаемого времени завершения,
- а также показывая все вышеперечисленное в одной постоянно обновляемой строке.

Решение всех этих проблем может потребовать больше времени и усилий разработчика, чем остальная часть содержимого цикла. Модуль tqdm решает все эти проблемы раз и навсегда, используя "*Pythonic*" шаблоны, для упрощения задач по добавления визуально привлекательных настраиваемых индикаторов выполнения без значительного снижения производительности даже в самых требовательных сценариях.

Для общих целей, модуль содержит режим интерфейса командной строки. CLI представляет собой полезный инструмент для системных администраторов, контролирующих поток данных по каналам.

Установка модуля tqdm в виртуальное окружение.

Модуль tqdm размещен на PyPI, поэтому установка относительно проста.

```
# создаем виртуальное окружение, если нет
$ python3 -m venv .venv --prompt VirtualEnv
# активируем виртуальное окружение
$ source .venv/bin/activate
# обновляем `pip`
(VirtualEnv):~$ python3 -m pip install -U pip
# ставим модуль `tqdm`
(VirtualEnv):~$ python3 -m pip install -U tqdm
```

Вверх

Использование модуля tqdm.

Модуль tqdm очень универсален и может использоваться разными способами. Основное и оригинальное использование tqdm это оборачивание итерируемых объектов Python. Просто оберните итерируемый объект классом tqdm.tqdm():

```
>>> from tqdm import tqdm
>>> from time import sleep
# оборачиваем итератор range(100) классом tqdm()
>>> for i in tqdm(range(100), ncols=80, ascii=True, desc='Total'):
...     sleep(0.1)
# Total: 100%|##################################| 100/100 [00:10<00:00,  9.91it/s]
```

Если во время выполнения нужен дополнительный вывод, то вместо функции print() используйте метод tqdm.write() и он предотвратит перенос прогресс бара:

```
>>> from tqdm import tqdm
>>> from time import sleep
>>> text = ""
>>> for char in tqdm(["a", "b", "c", "d"], ncols=80):
...     sleep(0.25)
...     text = text + char
...     tqdm.write(text)

# a
# ab
# abc
# abcd
# 100%|████████████████████████████| 4/4 [00:01<00:00,  3.97it/s]
```

Класс tqdm.trange(i) - это специальный оптимизированный экземпляр tqdm.tqdm(range(i)). Аргументы, которые он принимает при инициализации, аналогичны конструктора класса tqdm.tqdm():

```
>>> from tqdm import trange
>>> from time import sleep
>>> for i in trange(100, ncols=80, desc='Total'):
...     sleep(0.01)

# Total: 100%|████████████████████████████| 100/100 [00:01<00:00, 97.15it/s]
```

Создание экземпляра tqdm.tqdm() вне цикла позволяет вручную управлять строкой progressbar и добавлять динамические/обновляемые данные:

```
>>> from tqdm import tqdm
>>> from time import sleep
>>> pbar = tqdm(["a", "b", "c", "d"], ncols=80)
>>> for char in pbar:
...     sleep(0.25)
...     # добавление префикса и элементов итерации к прогресс бару
...     pbar.set_description(f"Processing '{char}'")

# Processing 'd': 100%|████████████████████████████| 4/4 [00:01<00:00,  3.97it/s]
```

Модуль tqdm поддерживает вложенные индикаторы выполнения. Смотрим пример:

```
>>> from tqdm.auto import trange
>>> from time import sleep
>>> for i in trange(4, desc='loop-1'):
...     for j in trange(5, desc='loop-2'):
...         for k in trange(50, desc='loop-3', leave=False):
...             sleep(0.01)
```

Ручное управление обновлениями строки прогресс бара класса tqdm() с помощью оператора with (могут быть проблемы с обновлением):

```
>>> from time import sleep
>>> from tqdm import tqdm
>>> with tqdm(total=30) as pbar:
...     for i in range(3):
```

```
...         sleep(0.1)
...         pbar.update(10)
```

Добавление описания и дополнительной статистики к progressbar.

Пользовательская информация может отображаться и динамически обновляться на прогресс барах модуля tqdm с помощью аргументов desc и postfix:

```
from tqdm import tqdm, trange
from random import random, randint
from time import sleep

with trange(10) as t:
    for i in t:
        # Описание будет отображаться слева
        t.set_description('GEN %i' % i)
        # postfix будет отображаться справа,
        # форматируется автоматически на основе типа данных аргумента
        t.set_postfix(loss=random(), gen=randint(1,999), str='h', lst=[1, 2])
        sleep(0.1)

with tqdm(total=10, bar_format="{postfix[0]} {postfix[1][value]:>8.2g}",
          postfix=["Batch", dict(value=0)]) as t:
    for i in range(10):
        sleep(0.1)
        t.postfix[1]["value"] = i / 2
        t.update()
```

Что нужно помнить при использовании {postfix[...]} в аргументе bar_format конструктора tqdm:

- в postfix также необходимо передать в качестве начального аргумента в совместимом формате
- и postfix будет автоматически преобразован в строку, если это объект типа dict. Чтобы предотвратить такое поведение, вставьте в словарь дополнительный элемент, где ключ не является строкой.

Дополнительные параметры bar_format также могут быть определены путем переопределения format_dict, а сама панель может быть изменена с помощью ascii:

```
from tqdm import tqdm

class TqdmExtraFormat(tqdm):
    """Предоставляет параметр формата `total_time`"""
    @property
    def format_dict(self):
        d = super(TqdmExtraFormat, self).format_dict
        total_time = d["elapsed"] * (d["total"] or 0) / max(d["n"], 1)
        d.update(total_time=self.format_interval(total_time) + " in total")
        return d

for i in TqdmExtraFormat(range(9), ascii=" .oO0",
      bar_format="{total_time}: {percentage:.0f}%|{bar}{r_bar}"):
    if i == 4:
        break

# 00:00 in total: 44%|0000.     | 4/9 [00:00<00:00, 962.93it/s]
```

Обратите внимание, что {bar} также поддерживает спецификатор формата [width][type].

- width:
  - по умолчанию - не указано и автоматически заполняет ncols (всю строку экрана);
  - int >= 0: фиксированная ширина переопределяет аргумент ncols;
  - int < 0: вычитает из автоматического значения по умолчанию (т.е. от длины строки экрана)
- type:
  - a: ascii (переопределяет аргумент как ascii=True)
  - u: unicode (переопределяет аргумент как ascii=False)
  - b: пусто (переопределяет аргумент как ascii=' ')

Это означает, что фиксированная полоса с текстом, выровненным по правому краю, может быть создана с помощью: bar_format='{l_bar}{bar:10}|{bar:-10b}right-justified'.

Обертывание методов чтения/записи.

Чтобы пропускать изменения в progressbar через методы файлового объекта file.read() или file.write(), можно использовать метод модуля tqdm.wrapattr():

Смотрим пример на основе скачивания файла по URL-адресу:

```
import requests, os
from tqdm import tqdm

eg_link = "https://caspersci.uk.to/matryoshka.zip"
response = requests.get(eg_link, stream=True)
with tqdm.wrapattr(open(os.devnull, "wb"), "write",
                   miniters=1, desc=eg_link.split('/')[-1],
                   total=int(response.headers.get('content-length', 0))) as fout:
    for chunk in response.iter_content(chunk_size=4096):
        fout.write(chunk)
```

Известные проблемы модуля tqdm.

Наиболее распространенные проблемы связаны с чрезмерным выводом в несколько строк вместо аккуратного однострочного индикатора выполнения.

- Консоли в целом: требуется поддержка возврата каретки (CR, r).
- Вложенные индикаторы выполнения:
  - Консоли в целом: требуется поддержка перемещения курсора вверх на предыдущую строку. Например, IDLE, ConEmu и PyCharm не имеют полной поддержки.
  - Windows: дополнительно может потребоваться модуль colorama, чтобы вложенные столбцы оставались в пределах соответствующих строк.
- Юникод:
  - Среды, которые поддерживают Unicode, будут иметь сплошные гладкие индикаторы выполнения. Запасной вариант - это полоса, состоящая только из символов ascii.
  - Консоли Windows часто лишь частично поддерживают Unicode и поэтому часто требуют явной передачи аргумента ascii=True. Это связано либо с тем, что символы Юникода нормальной ширины неправильно отображаются, либо с тем, что некоторые символы Юникода вообще не отображаются.
- Встроенные функции генераторы Python:
  - Встроенные функции генераторы Python имеют тенденцию скрывать длину итерируемых объектов.
  - Замените tqdm(enumerate(...)) на enumerate(tqdm(...)) или tqdm(enumerate(x), total=len(x),...). То же самое относится к numpy.ndenumerate.
  - Замените tqdm(zip(a, b)) на zip(tqdm(a), b) или даже zip(tqdm(a), tqdm(b)).
  - То же самое относится и к встроенному модулю itertools.

Удобные функции модуля tqdm.

Для более удобной работы по построению прогресс баров, подмодуль tqdm.contrib содержит несколько оптимизированных функций, эквивалентных встроенным.

*tqdm.contrib.tenumerate(iterable, start=0, total=None, tqdm_class=tqdm.auto.tqdm, **tqdm_kwargs)*:

Функция tqdm.contrib.tenumerate() это эквивалент numpy.ndenumerate или встроенной функции enumerate().

- **tqdm_kwargs: ключевые аргументы для настройки класса tqdm.tqdm().

*tqdm.contrib.tzip(iter1, *iter2plus, **tqdm_kwargs)*:

Функция tqdm.contrib.tzip() это эквивалент встроенной функции zip().

- **tqdm_kwargs: ключевые аргументы для настройки класса tqdm.tqdm().

*tqdm.contrib.tmap(function, *sequences, **tqdm_kwargs)*:

Функция tqdm.contrib.tmap() это эквивалент встроенной функции map().

- **tqdm_kwargs: ключевые аргументы для настройки класса tqdm.tqdm().

*tqdm.contrib.itertools.product(*iterables, **tqdm_kwargs)*:

Фу Вверх qdm.contrib.itertools.product() это эквивалент функции itertools.product().

- **tqdm_kwargs: ключевые аргументы для настройки класса tqdm.tqdm().

*tqdm.contrib.telegram.tqdm(*iterables, token=None, chat_id=None, **tqdm_kwargs)*:

Отправляет обновления боту Telegram. Аргумент **tqdm_kwargs это ключевые аргументы класса tqdm.tqdm().

```
from tqdm.contrib.telegram import tqdm, trange
for i in tqdm(iterable, token='{token}', chat_id='{chat_id}'):
    ...
```

tqdm.contrib.logging.logging_redirect_tqdm(loggers=None, tqdm_class=std_tqdm):

logging_redirect_tqdm() это вспомогательная функциональность для взаимодействия с ведением журнала stdlib. Диспетчер контекста перенаправляет ведение журнала в консоли на tqdm.write(), не затрагивая другие обработчики ведения журнала (например, файлы журнала).

- Необязательный аргумент loggers: это список обработчиков, требующих перенаправления (по умолчанию: [logging.root]).
- Необязательный аргумент tqdm_class: класс модуля tqdm, который будет заниматься перенаправлением.

Пример использования:

```
import logging
from tqdm import trange
from tqdm.contrib.logging import logging_redirect_tqdm

LOG = logging.getLogger(__name__)

if __name__ == '__main__':
    logging.basicConfig(level=logging.INFO)
    with logging_redirect_tqdm():
        for i in trange(9):
            if i == 4:
                LOG.info("Ведение журнала консоли перенаправлено на `tqdm.write()`")
    # logging restored
```

tqdm.contrib.concurrent:

Подмодуль tqdm.contrib.concurrent это тонкая обертка вокруг встроенного модуля concurrent.futures.

Подмодуль включает в себя насколько функций:

- ensure_lock(tqdm_class, lock_name=""): функция получает или создает при необходимости, а затем восстанавливает блокировку tqdm_class.
- thread_map(fn, *iterables, **tqdm_kwargs): эквивалент list(map(fn, *iterables)), управляемый ThreadPoolExecutor.

  Принимаемые аргументы:

  - необязательный аргумент max_workers: максимальное количество рабочих потоков; передается в ThreadPoolExecutor.
  - **tqdm_kwargs: это ключевые аргументы для настройки tqdm.tqdm()
- process_map(fn, *iterables, **tqdm_kwargs): эквивалент list(map(fn, *iterables)), управляемый ProcessPoolExecutor.

  Принимаемые аргументы:

  - необязательный аргумент max_workers: максимальное количество рабочих процессов; передается в ProcessPoolExecutor.
  - необязательный аргумент chunksize: размер блоков, отправляемых рабочим процессам; передается в ProcessPoolExecutor.map(). По умолчанию 1.
  - lock_name это член tqdm_class.get_lock() для использования, по умолчанию: mp_lock.
  - **tqdm_kwargs: это ключевые аргументы для настройки tqdm.tqdm()

Интеграция с модулем asyncio.

Обратите внимание, что оператор break в настоящее время не перехватывается асинхронными итераторами. Это означает, что в этом случае tqdm не может очистить себя после выхода из цикла. Для очистки ресурсов необходимо вручную закрыть pbar.close(), либо использовать синтаксис диспетчера контекста:

```
from tqdm.asyncio import tqdm

wi (range(9)) as pbar:
    for i in pbar:
```

Вверх

```
        if i == 2:
            break
```

Подмодуль tqdm.asyncio включает в себя еще несколько функций:

- tqdm_asyncio(): это асинхронная версия класса tqdm() (Python 3.6+).
- as_completed(cls, fs,*, loop=None, timeout=None, total=None, **tqdm_kwargs): это обертка для asyncio.as_completed.
- gather(cls, *fs, loop=None, timeout=None, total=None, **tqdm_kwargs): это обертка для asyncio.gather.
- tarange(*args, **kwargs): ссылка на tqdm.asyncio.tqdm(range(*args), **kwargs).

Интеграция с модулем Pandas.

Пример для DataFrame.progress_apply и DataFrameGroupBy.progress_apply:

```
import pandas as pd
import numpy as np
from tqdm import tqdm

df = pd.DataFrame(np.random.randint(0, 100, (100000, 6)))

# регистрация `pandas.progress_apply` и `pandas.Series.map_apply` с `tqdm`
# (можно использовать необязательные `kwargs` класса `tqdm.notebook.tqdm`)
tqdm.pandas(desc="my bar!")

# Теперь можно использовать `progress_apply` вместо `apply`
# и `progress_map` вместо `map`
df.progress_apply(lambda x: x**2)
# также можно группировать:
# df.groupby(0).progress_apply(lambda x: x**2)
```

Примечание: код выше будет работать, если модуль tqdm >= v4.8: для версий tqdm ниже 4.8 вместо tqdm.pandas() нужно сделать следующим образом:

```
from tqdm import tqdm, tqdm_pandas
tqdm_pandas(tqdm())
```

Если интересно, как модифицировать его для собственных обратных вызовов, импортируйте модуль и запустите help().

Интеграция с модулем IPython/Jupyter.

IPython/Jupyter поддерживается при помощи подмодуля tqdm.notebook. Пример для запуска в ноутбуке:

```
from tqdm.notebook import trange, tqdm
from time import sleep

for i in trange(3, desc='1st loop'):
    for j in tqdm(range(100), desc='2nd loop'):
        sleep(0.01)
```

Версия для ноутбука поддерживает проценты или пиксели для общей ширины (например: ncols='100%' или ncols='480px').

Также tqdm может автоматически выбирать между консольной версией или версией для ноутбука с помощью подмодуля tqdm.autonotebook:

```
from tqdm.autonotebook import tqdm
tqdm.pandas()
```

Обратите внимание, что при запуске в ноутбуке это может вызвать предупреждение TqdmExperimentalWarning, так как невозможно различить блокнот jupyter и консоль jupyter. Чтобы отключить это предупреждение необходимо использовать подмодуль tqdm.auto вместо tqdm.autonotebook.

Обратите внимание, что в блокнотах, полоса будет отображаться в ячейке, где он был создан. Это может быть не та ячейка, где она используется. Если это нежелательно, то:

- отложить создание прогресс бара до ячейки, где он должен отображаться;
- создать панель с display=False, а в более позднем вызове ячейки display(bar.container).

```
from tqdm.notebook import tqdm
pb = tqdm(..., display=False)
```

```
# другая ячейка
display(pbar.container)
```

Другая возможность состоит в том, чтобы иметь один прогресс бар (в верхней части ноутбука), который постоянно используется повторно (для этого можно использовать метод pbar.reset(), вместо pbar.close()). По этой причине версия tqdm для ноутбука, при возникновении исключения, не вызывает автоматически pbar.close().

```
from tqdm.notebook import tqdm
pbar = tqdm()

# другая ячейка
iterable = range(100)
# инициализация с новым `total`
pbar.reset(total=len(iterable))
for i in iterable:
    pbar.update()
# принудительно выводит окончательный статус, но не закрывает
pbar.refresh()
```

Режим интерфейса командной строки модуля tqdm.

```
$ seq 9999999 | tqdm --total 9999999 --ncols 80 | wc -l
100%|████████████████████████████████| 9999999/9999999 [00:01<00:00, 6088208.15it/s]
```

Резервное копирование большого каталога? Обратите внимание, что в качестве параметров командной строки можно использовать аргументы конструктора класса tqdm.tqdm().

```
$ tar -zcf - docs/ | tqdm --bytes --total `du -sb docs/ | cut -f1` \
  > ~/backup.tgz
 44%|██████████████                  | 153M/352M [00:14<00:18, 11.0MB/s]
```

А можно сделать и так:

```
$ BYTES="$(du -sb docs/ | cut -f1)"
$ tar -cf - docs/ \
  | tqdm --bytes --total "$BYTES" --desc Processing | gzip \
  | tqdm --bytes --total "$BYTES" --desc Compressed --position 1 \
  > ~/backup.tgz
Processing: 100%|████████████████████| 352M/352M [00:14<00:00, 30.2MB/s]
Compressed:  42%|████████            | 148M/352M [00:14<00:19, 10.9MB/s]
```

Параметры командной строки модуля tqdm.

- delim : символ-разделитель, по умолчанию: '\n'. NB: в системах Windows Python преобразует '\n' в '\r\n'.
- buf_size: целое число, размер строкового буфера в байтах (по умолчанию: 256), используемый при указании разделителя.
- bytes : bool, если true, то будут подсчитываться байты, игнорируются разделители, а также по умолчанию для unit_scale будет установлено значение true, для unit_divisor - 1024, а для unit - 'B'.
- tee: bool, если true, то передает stdin как в stderr, так и в stdout.
- update: bool, если true, то будет обрабатывать входные данные как недавно прошедшие итерации, т. е. числа для передачи в update(). Обратите внимание, что это происходит медленно (~2e5 it/s), т.к. каждый ввод должен быть декодирован как число.
- update_to: bool, если true, будет обрабатывать входные данные как общее количество прошедших итераций, т. е. числа для присвоения self.n. Обратите внимание, что это происходит медленно (~2e5 it/s).
- null: bool, если true, то ввод будет отброшен (без стандартного вывода).
- manpath: str, каталог для справочных страниц tqdm.
- comppath: строка, каталог в котором стоит завершить работу tqdm.
- log: строка, CRITICAL|FATAL|ERROR|WARN(ING)|[default: 'INFO']|DEBUG|NOTSET.

Примечание: Обратите внимание, что в качестве параметров командной строки можно использовать аргументы конструктора класса tqdm.tqdm().

Содержание раздела:

Вверх

Вверх