Creating custom data generator for training Deep Learning Models-Part 1

Anuj shah (Exploring Neurons) Jul 9, 2019 · 6 min read

This series will tell you how you can create your own data generator function for training deep learning models. Using a data generator to load data is imperative while training deep learning models as we need large dataset to train good models, and large datasets requires large memory space, that can be a bottleneck and this is where generator functions come to our rescue. Writing your own generator function to load the data is not that tough but it can be quite cumbersome if you are not very clear conceptually. This posts tries to elucidate step by step, how you can achieve this and will give you clear understanding of how you can do this for different kind of inputs to be fed into the network.

Note: Although I will use keras in this series to show you how to load the data created through our generator functions but the concept to create generator function can be used to load data to any other libraries like TensorFlow, PyTorch, etc.

This series goes as follow:

1. **What is a generator function in python and the difference between yield and return.**
2. **Writing a generator function to read your data that can be fed for training an image classifier in Keras.**
3. **Creating different generator to read different kinds of inputs (sequence of inputs, multiple inputs) for training NN model in Keras.**

Chapter -1 : What is a generator function in python and the difference between yield and return

I have followed number of sources online to understand and write this post, I will give the references and credits as I go, or at the end, and if I missed any, kindly let me know and I will add it.

Let’s get started by first understanding what a generator function is and how it works. This will also elucidate the difference between yield and return keyword-

Let's say you and your friends go to a restaurant and order some starter, main course, soups and desserts. we have two waiters — a normal python function and a generator function. what happens next

waiter-1-normal python function.

waiter-1 gets everything all at once and put it on your table, this may fill up all the space on our table and it may be difficult for you to eat.

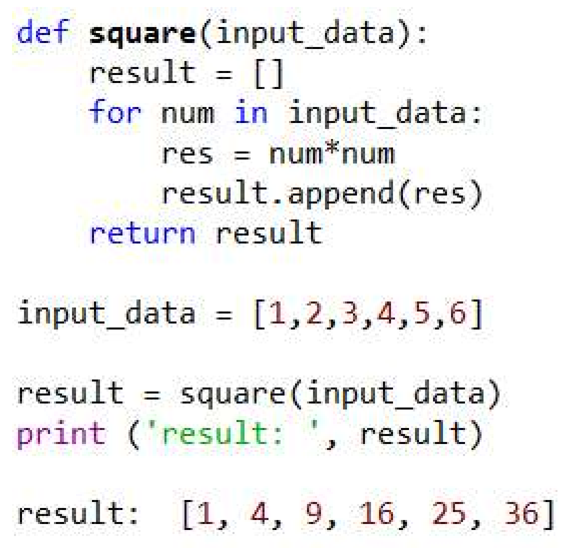
waiter-1 (normal function) gets everything all at once

waiter-2 generator function

waiter-2 gets your order too but it does so one by one, It fetches you the first order and then next and then the next and so on, it remembers which dishes have been served and always fetch the next one.

waiter-2 (generator function) gets your order one by one.

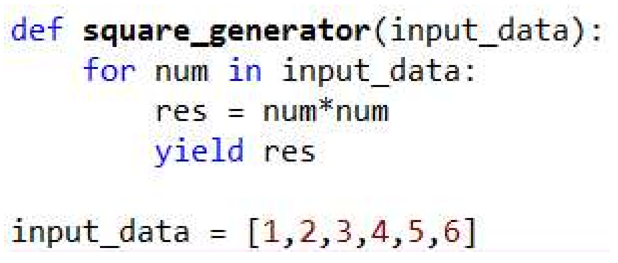
I hope you got the gist of it and now lets get technical — Lets say we have to write a function that takes a list of number as input and returns the square of every element. Let's write using normal function



```
def square(input_data):
    result = []
    for num in input_data:
        res = num*num
        result.append(res)
    return result

input_data = [1,2,3,4,5,6]

result = square(input_data)
print ('result: ', result)

result:  [1, 4, 9, 16, 25, 36]
```

Its quite simple function, a list result is created inside the function which holds the square of every element and get returned at end .

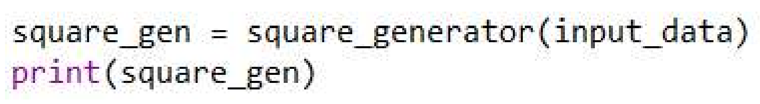
Lets write a generator function for the same logic



```
def square_generator(input_data):
    for num in input_data:
        res = num*num
        yield res

input_data = [1,2,3,4,5,6]
```

Now if we observe the generator function, it does not require any list to store the result and the ***return*** keyword is replaced by ***yield*** keyword.

Now lets call this generator function and see what it prints

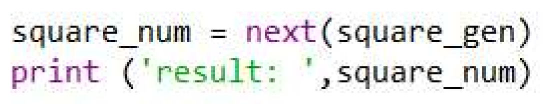


```
square_gen = square_generator(input_data)
print(square_gen)
```

> *<generator object square_generator at 0x0000000008B82990>*

it prints that square gen is a generator object of square generator. In order to get the output from generator function we can use python built in iterator — the ***next*** keyword

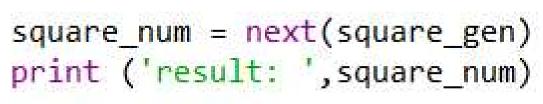
to get the first element



```
square_num = next(square_gen)
print ('result: ',square_num)
```

> *result: 1*

to get second element you use the next again

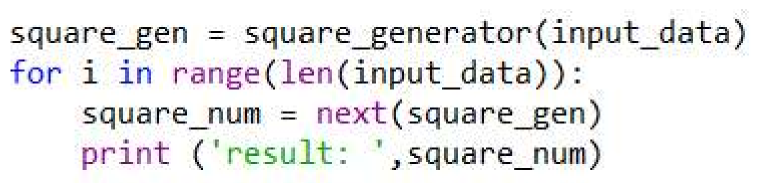


```
square_num = next(square_gen)
print ('result: ',square_num)
```

> *result: 4*

and similarly the third element and so on..

you can also put this in a loop:



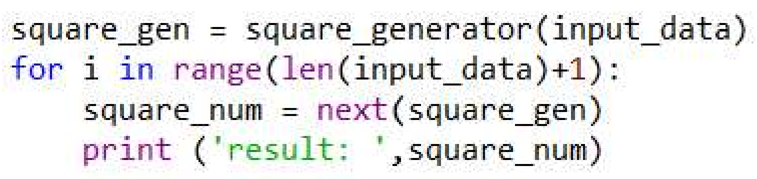
```
square_gen = square_generator(input_data)
for i in range(len(input_data)):
    square_num = next(square_gen)
    print ('result: ',square_num)
```

> *result: 1*
> *result: 4*
> *result: 9*
> *result: 16*

```
result: 25
result: 36
```

Now if you observe carefully I reinitialize the generator object or else it would have printed from third result onward as we already called the next function twice previously and generator resumes the operation form where it has left.

Lets run the loop for more times than there is element in the input data.



```python
square_gen = square_generator(input_data)
for i in range(len(input_data)+1):
    square_num = next(square_gen)
    print ('result: ',square_num)
```

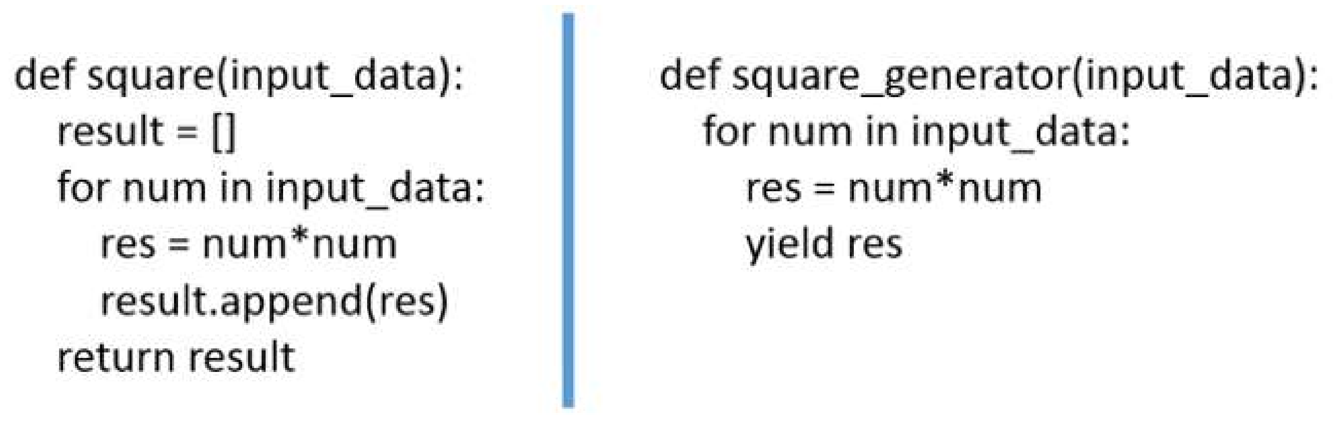
The output we get is

```
result: 1
result: 4
result: 9
result: 16
result: 25
result: 36
Traceback (most recent call last):

File "<ipython-input-27-a3bcf4dd17d3>", line 3, in <module>
square_num = next(square_gen)

StopIteration
```

As there are only 6 elements in input data , after printing 6 outputs, it stops giving a StopIteration error. we can put this in a ***try-except*** block to catch the error and we can either stop it or reinitialize it to loop again by defining some logic.

Now that we have seen the example

Let's understand the difference between normal and generator function , return and yield keyword.



```python
def square(input_data):
    result = []
    for num in input_data:
        res = num*num
        result.append(res)
    return result
```

```python
def square_generator(input_data):
    for num in input_data:
        res = num*num
        yield res
```

In the generator function, there is yield keyword rather than return. And now we can define the generator function as —

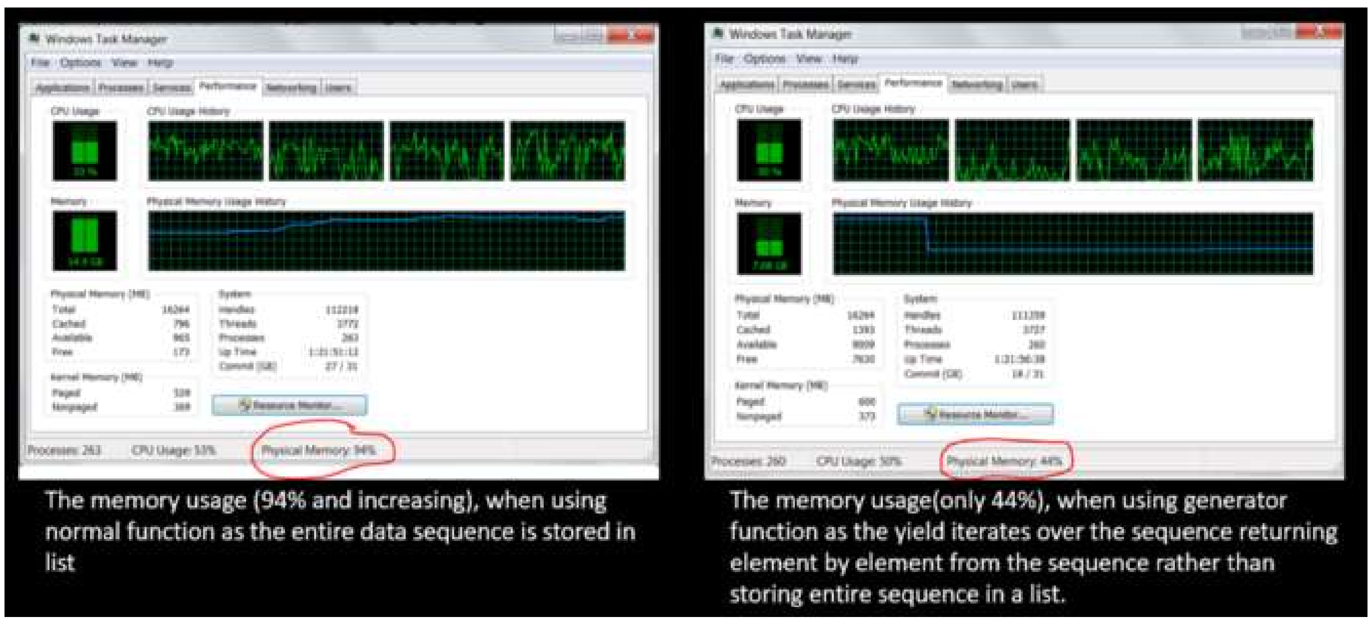
> *"A generator function is defined like a normal function, but whenever it needs to generate a value, it does so with the yield keyword rather than return. If the body of a def contains yield, the function automatically becomes a generator function."*
>
> *"Generator are functions that yield result, suspend the operation and then resume the operation."*

Lets highlight some more points :

> ***1. Return*** *sends a specified value back to its caller whereas* ***Yield*** *can produce a sequence of values.*
>
> ***2.*** *Return basically exits from a function with a value and the local values within it is destroyed whereas yield is sort of exiting from the function but it remembers the state when it exits. When you do* ***next*** *from your function, it will resume from the point where it yielded.*
>
> ***3.****The yield statement suspends function's execution and sends a value back to caller, but retains enough state to enable function to resume where it is left off. When resumed, the function continues execution immediately after the last yield run. This allows its code to produce a series of values over time, rather them computing them all at once and sending them back like a list.*
>
> ***4.*** *We should use yield when we want to iterate over a sequence, but don't want to store the entire sequence in memory.*

This video by Corey Schafer further elucidates the concept very clearly — Python Tutorial: Generators — How to use them and the benefits you receive.

The image below illustrates the memory usage when using normal function vs using data generator for training a neural network :



The memory usage (94% and increasing), when using normal function as the entire data sequence is stored in list

The memory usage(only 44%), when using generator function as the yield iterates over the sequence returning element by element from the sequence rather than storing entire sequence in a list.

memory usage- left: 94% and increasing when using normal function; right: 44% when using generator function

Credits:

Corey Schafer — Python Tutorial: Generators — How to use them and the benefits you receive

GeeksforGeeks — https://www.geeksforgeeks.org/generators-in-python/

GeeksforGeeks — https://www.geeksforgeeks.org/use-yield-keyword-instead-return-keyword-python/

I hope this post makes you understand, what a generator function is and it's importance specially when we need to load or use large amount of data. In the next post we will use them to generate data to be fed in to Neural networks for training a image classifier and you can appreciate them in action.

Till then, Keep Learning!! Keep Exploring Neurons!!!!

Python Data Generator Custom Data Generator Keras Deep Learning