parallel_function

April 27, 2020

1 Parallel processing in Python

1.1 Import libraries

[1]:

```
import numpy as np
import time
from joblib import Parallel, delayed
```

1.2 Mimic something to be run in parallel

Here we will create a function that takes 2 seconds to run, to mimic a model or a complex function.

[2]:

```
def my_slow_function(x):
    """A very slow function, which takes 1 second to double a number"""

    # A 2 second delay
    time.sleep (1)

    return x * 2
```

1.3 Running our function sequentially in a for loop

[3]:

```
# Get start time
start = time.time()
# Run functions 8 times with different input (using a list comprehension)
trial_output = [my_slow_function(i) for i in range(8)]
print(trial_output)
# Get time taken
time_taken = time.time() - start
# Print time taken
print (f'Time taken: {time_taken:.1f} seconds')
```

```
[0, 2, 4, 6, 8, 10, 12, 14]
Time taken: 8.0 seconds
```

That's a good improvement in speed!

1.4 Running our function in parallel using joblib

`n_jobs` is the maximum number of CPU cores to use. If set to -1, all available cores will be used.

[4]:

```python
# Get start time
start = time.time()
# Run functions 8 times with different input using joblib
trial_output = \
    Parallel(n_jobs=-1)(delayed(my_slow_function)(i) for i in range(8))
print(trial_output)
# Get time taken
time_taken = time.time() - start
# Print time taken
print (f'Time taken: {time_taken:.1f} seconds')
```

```
[0, 2, 4, 6, 8, 10, 12, 14]
Time taken: 1.3 seconds
```

1.5 Checking pseudo-random number generation

Pseudo-random number generators, if not provided with a seed, use the computer clock to generate the seed. This means some methods of parallel processing will generate sets of random numbers that may be the same. By default joblib uses the loki backend which prevents this occurring, but let's check.

[5]:

```python
def numpy_random():
    """Generate a random number using NumPy"""
    return np.random.random()
```

[6]:

```python
Parallel(n_jobs=-1)(delayed(numpy_random)() for i in range(5))
```

[6]:

```
[0.1150670010400634,
 0.7929535912044415,
 0.24692567519542463,
 0.08106962217687963,
 0.6722579989206016]
```