DD2480
Group 15

Report

Assignment 3

Project

Name: PyTensor

URL: https://github.com/pymc-devs/pytensor

PyTensor is a Python library that defines, optimises, and efficiently evaluates mathematical expressions involving multi-dimensional arrays.

Lines of code: 69 216

Branch coverage: 81%

Onboarding experience

Did it build and run as documented?

The documentation was easy to follow. It was recommended to set up a conda environment using a given .yml file which was easy to do. However, in the next step, the documentation referred to a file which was not included in the repository, more specifically `requirements.txt`. The project still seems to run as expected without the requirements which leads us to believe that the documentation is simply a bit outdated.

Complexity

1. What are your results for five complex functions?

Function #	Function Location	Cyclomatic Complexity
1	_debugprint@464-740@./printing.py	57
2	import_node@318-380@.\graph\fg.py	13
3	perform@211-250@.\tensor\blas.py	12
4	mean@1588-1674@.\tensor\math.py	19
5	filter@141-262@.\tensor\type.py	26

1.1. Did all methods (tools vs. manual count) get the same result?

- **_debugprint@464-740@./printing.py**:
  - Using lizard: 57
  - Counting manually (Love): 58 (53 if statements, 2 for loops, 3 return statements).
- **import_node@318-380@.\graph\fg.py**
  - Yes, 13 / 14 depending on if you include exceptions (M = P + 1, where P is the number of predicate nodes (5 if, 2 and, 5 for, 1 assert, 1 raise))
- **perform@211-250@.\tensor\blas.py**
  - Yes: 12 / 13 depending on if you include exceptions (M = P + 1, where P is the number of predicate nodes (6 if, 4 and, 1 or, 1 raise))
- **mean@1588-1674@.\tensor\math.py**
  - Yes: 19 / 21 depending on if you include exceptions (M = P + 1, where P is the number of predicate nodes (10 if, 2 elif, 3 and, 1 or, 1 for, 1 early return, 2 raise))
- **filter@141-262@.\tensor\type.py**
  - Yes: 26 / 35 depending on if you include exceptions (M = P + 1, where P is the number of predicate nodes (16 if, 3 elif, 5 and, 1 for, 9 raise))

1.2. Are the results clear?

The results are clear, our calculations and the result from running *lizard* are the same.

2. Are the functions just complex, or also long?

- **_debugprint@464-740@./printing.py**: The function consists mostly of if statements that are merely updating and confirming the values of variables. This leads to the function being longer, but not necessarily more complex than one would think, given the presented CC.
- **import_node@318-380@.\graph\fg.py:** The function is complex but short. The function includes multiple for-loops within for-loops and if-statements within if-statements within these double for-loops. The function itself is only around 25 LOC.
- **perform@211-250@.\tensor\blas.py:** The function contains if statements using and which increases its complexity. Otherwise, it is a short function.
- **mean@1588-1674@.\tensor\math.py:** The function contains a lot of if and elif statements which increases its complexity. The function is fairly long.
- **filter@141-262@.\tensor\type.py:** The function contains a lot of if and elif statements which increases its complexity. The function is fairly long.

3. What is the purpose of the functions? Is it related to the high CC?

- **_debugprint@464-740@./printing.py**: Prints a graph. The function does several checks on its many arguments which increases its CC.
- **import_node@318-380@.\graph\fg.py:** The purpose of the function is to recursively import nodes from a graph, starting at a specific node. Because the function is handling a graph, it is reasonable that the function has a high CC.
- **perform@211-250@.\tensor\blas.py:** This function performs a matrix multiplication and vector addition using the formula "beta * y + alpha * A x". Matrix multiplication and vector addition requires for-loops which increase the function's CC.
- **mean@1588-1674@.\tensor\math.py:** Computes the mean value along a given axis(es) of a tensor. The function does several type checks which increase the function's CC.
- **filter@141-262@.\tensor\type.py:** This function validates and pre-processes input data to ensure compatibility with the rest of the code base, handles type conversion, and checks dimension, alignment and shape of data. Since there are so many checks, i.e. if statements, it is reasonable that the function has a high CC.

4. Are exceptions taken into account in the given measurements?

Exceptions are not taken into account in the given measurements. If we think of an exception as another possible branch, then we are adding additional paths that contribute to higher cyclomatic complexity.

5. Is the documentation clear w.r.t. all the possible outcomes?

- **_debugprint@464-740@./printing.py**: The documentation is clear but could be extended with more detail.
- **import_node@318-380@.\graph\fg.py:** The documentation is not clear w.r.t. the possible outcomes. The documentation does not state any of the possible outcomes.
- **perform@211-250@.\tensor\blas.py:** The documentation is not clear w.r.t. the possible outcomes. The documentation does not state any of the possible outcomes.
- **mean@1588-1674@.\tensor\math.py:** The documentation for this function is clear and detailed concerning the possible outcomes.
- **filter@141-262@.\tensor\type.py:** The documentation for this function is clear and detailed concerning the possible outcomes.

Refactoring

Plan for refactoring complex code:
Breaking up large functions into smaller ones is key to reducing complexity.

Use all() instead of multiple and statements where needed.

Use a dictionary instead of multiple if/elif. Requires one look-up instead of multiple comparisons.

Estimated impact of refactoring (lower CC, but other drawbacks?):
Only focusing on CC can lead to redundant and unnecessary code that is hard to understand.

~~**Carried out refactoring (optional, P+):**~~

Coverage

Tools

Document your experience in using a "new"/different coverage tool.

We used Coverage.py.

```
coverage run -m pytest test_file.py
```

then

```
coverage report
```

which showed us the coverage. It was easy to use.

How well was the tool documented? Was it possible/easy/difficult to integrate it with your build environment?

The tool was well documented and it was easy to integrate it with our build environment. Worked seamlessly with pytest.

Your own coverage tool

Show a patch (or link to a branch) that shows the instrumented code to gather coverage measurements.

branch: https://github.com/DD2480-Group-15-vt24/3-pytensor/tree/part-2-task-1
All functions above have been commented with an branch ID and is flagged.

What kinds of constructs does your tool support, and how accurate is its output?

Evaluation

1. How detailed is your coverage measurement?

The coverage measurement is very detailed in regard to the functions we implemented it for.

2. What are the limitations of your own tool?
Might give false positive/negatives
Does not capture the full behaviour of the program.
It is tailored to just one function, and therefore not very efficient.

3. Are the results of your tool consistent with existing coverage tools?
This is difficult to answer since Coverage.py does not work at a function level.

Coverage improvement

Show the comments that describe the requirements for the coverage.

Report of old coverage:
mean: 60%
filter: 66%

Report of new coverage:
mean: 80%
filter: 83%

Test cases added:

```
# New test 1, branch #69 --> 66,6666%
def test_dtype_arg():
    with pytest.raises(NotImplementedError):
        mean(np.zeros(1), mean_coverage, op=True, dtype="float32")


# New test 2, branch #70 --> 73,33333%
def test_acc_dtype_arg():
    with pytest.raises(NotImplementedError):
        mean(np.zeros(1), mean_coverage, op=True, acc_dtype="float32")

# New test 3, branch #71 --> 80%
def test_acc_dtype_arg_2():
    ll = [shared(0.0), shared(2.0)]
    assert(mean(ll, mean_coverage, axis=0, keepdims=True, op=True), TensorVariable)
```

```
# New test 4, branch #94, 95 and 96 true
def test_up_dtype():
    test_type = TensorType(config.floatX, shape=())
    data = np.array([1, 2, 3], dtype='int32')
    with pytest.raises(TypeError):
        test_type.filter(data, filter_covarage)
```

Self-assessment: Way of working

Current state according to the Essence standard: We are back at the "Foundation Established" state. We have picked the main practices and tools we need to work together, but we are not using them and our main issue is communication.

Was the self-assessment unanimous? Any doubts about certain items?
Yes

How have you improved so far?
Over all we have not improved, our communication has been very bad this week. One new thing that was nice was that Selma and Albin pair-programmed, and it worked nicely.

Where is potential for improvement?
Communication, we are very bad at communicating and all members are not showing up or doing work. The potential lies in that all members of the group should show up and take responsibility for the work, and all members should communicate better and not leve thing for the last minute.

Overall experience

What are your main take-aways from this project? What did you learn?
Overall we have learned many new things, how to calculate CC, how to use coverage tools, make new coverage tools and improve the branch coverage.

Is there something special you want to mention here?