



F. Wermelinger
Office: Pierce 211

Homework 4

Git rebase, Automatic differentiation and
First steps with continuous integration

Issued: October 11th, 2022 Due: October 25th, 2022

Note this is a 2 week exercise. *Do not procrastinate the work.*

Problem 1: Homework Submission Requirements (10 points)

Requirements:

1. Complete the homework on the designated branch (5 points)
2. Create a pull request to merge the designated homework branch into your default branch, e. g., main or master (2 points)
3. Merge the open pull request of the *previous homework* into your default branch **after** you have received feedback from the teaching staff and regrade requests are resolved (3 points).

Deliverables: your solutions to the problems below must be submitted in a directory called [submission](#).

See <https://harvard-iacs.github.io/2022-CS107/pages/tutorials.html#tutorial-hw>.

Problem 2: Interactive Git Rebase (10 points)

Deliverables:

1. [P2_rebase.png](#)
2. [P2_conflict.md](#)

In this problem you are going to practice an interactive rebase scenario. Recall that when you have local commits in your repository (not pushed to any shared remote yet) you can freely change and rebuild your commit history as you like.¹ This is useful to tidy up a local history before you publish it to your colleagues.

We continue with the same workflow as in the previous homework. You can find shell script in [code/p2/P2.sh](#) that will create the Git setup for you. The script will create a remote and local Git repository. Again, to *avoid nested* Git repositories, do not execute the script in your private repository but somewhere outside of it like shown in directory hierarchy as we have used it in previous explanations of previous homework.

```
classes/
|-- CS107
|   |-- git
|   |   |-- myrepo      <- your private Git repository
|   |   |-- sandbox    <- forked sandbox from earlier homework
|   |   |-- hw2        <- from earlier homework
|   |   |-- hw3        <- from last homework
|   |   |-- hw4
|   |   |-- P2.sh      <- copy of script from HW handout
|   |   |-- remote     <- remote repository for this problem
|   |   |-- \-- local  <- developer B local repository for this problem
|   |   \-- main
|   \-- CS107_other_data
\-- CS205 (some other class in this directory)
```

For this problem you will need to work in the local Git repository only (created by running the script `P2.sh`). If you did a mistake start fresh by executing the script `P2.sh` again (repositories local and remote will be removed and created again).

a) 5 points

Once you have setup the necessary Git repositories (see problem introduction above), change into the local Git repository. This repository is in a state you had before lunch and you want to clean up your history a little bit before you push your changes to the remote. Inspect the history with `git log` to orient yourself. Be sure to understand which commits are *local* by finding the commit that is currently the head on the branch `origin/main`. All the commits on top of this commit are still local to your repository only. Your history should look similar to

¹This applies only to local commits not pushed to any shared remote of course. Any commit hashes already pushed can not be changed anymore.

```
$ git log --oneline
bf005f0 (HEAD -> main) Extending new feature with component B
78df141 Extending new feature with component A
639a1a9 Adding new feature to code
8ffd084 Minor (bad commit message will reword later)
5a98938 Forgot to add some code (will fixup later)
0c87cc9 Some more changes added to file
47df47f (origin/main) Initial
```

You can see there are a few commits in the history that need some cleaning up. You will do this with the command `git rebase --interactive`. This will open up your editor where you can modify your current local history. Read through the command documentation available for the interactive rebase (you can find them as comments in the file that was opened). Note that the commit list in an interactive rebase is oldest local commit on top and newest at the bottom (`git log` will show them to you in the opposite order). Perform the following tasks with the interactive rebase:

1. Fixup the commit with the message “Forgot to add some code (will fixup later)”. A fixup means to take the changes in that commit and apply them in the previous commit. Use the commit message of the previous commit.
2. Reword the commit with the message “Minor (bad commit message will reword later)”. You may inspect the changes introduced in this commit using `git show <commit id>` and come up with a more meaningful commit message.
3. Drop the commit with the message “Extending new feature with component B”. You figured that component B in the new feature is no longer needed.

You can either perform all these in one interactive session or you can apply these changes one at a time in different interactive rebase sessions. To apply a change, save the file you are editing during the interactive rebase and close the editor. You can use `git log` to inspect the new local history afterwards. Your new local history should look similar to:

```
$ git log --oneline
cfe0b71 (HEAD -> main) Extending new feature with component A
4f081df Adding new feature to code
72f6458 Your commit message may be more enthusiastic than this one
4109012 Some more changes added to file
47df47f (origin/main) Initial
```

Take a screenshot of your new history and save it in the file [P2_rebase.png](#). Do not push your local history to the remote.

b) 5 points

You did not push your local history yet because there was some uncertainty about the new feature added in the commit with the message “Adding new feature to code”. After a meeting with your colleagues, you have decided to remove this feature all

together. Open another interactive rebase session and drop the commit with the message “Adding new feature to code”. This should result in a merge conflict. Please explain why this conflict happened (*you do not need to resolve it*). Include the following considerations in your argument:

- What about the child commit of the commit you just dropped? (2 *points*)
- How could you avoid this merge conflict? (1 *points*)
- What if the child commit contained other changes not related to the new feature? What would be a good commit strategy to make your interactive rebase life easier? (2 *points*)

Write your explanation in the file `P2_conflict.md` using at most 5 sentences.

Problem 3: Motivating Automatic Differentiation (20 points)

Deliverables:

1. [P3.py](#)
2. [P3.png](#)

Important: this problem will be partially *auto-graded*. You are expected to work with the skeleton code provided in [code/p3/P3.py](#). Do not change existing code in that skeleton code. Only add your solution code in appropriate places or update values in existing variables. Otherwise you will risk that the auto-grader fails and you lose points. Read carefully the problem statement and ensure your implementation follows the specifications given.

For scalar functions of a single variable, the derivative is defined by

$$\frac{df}{dx} = f'(x) = \lim_{\epsilon \rightarrow 0} \frac{f(x + \epsilon) - f(x)}{\epsilon} \quad (1)$$

In the following we will approximate this derivative using a finite but small value for ϵ and perform some experiments with analytic expressions.

a) **7 points**

Write an outer function called `approximate` which takes as inputs a function of a single variable `f` and a value `eps` for ϵ (in that order). This function returns a closure that expects an argument `x` for input and shall compute the first-order finite difference approximation to the first derivative of $f(x)$ as shown in equation 1. The value `x` may be a single number or a NumPy array of numbers. Start with the provided skeleton file [code/p3/P3.py](#)

b) **8 points**

Let $f(x) = \ln(x)$ and let $f'(x)$ denote the first derivative of $f(x)$. Use your closure from task 3a to compute the approximation $f'_{FD}(x)$ for $0.2 \leq x \leq 0.4$ and $\epsilon = 1 \times 10^{-1}$, $\epsilon = 1 \times 10^{-7}$ and $\epsilon = 1 \times 10^{-15}$. Create one plot and compare $f'_{FD}(x)$ at the different values of ϵ to the exact derivative. Make sure you label the axes. Save your plot as [P3.png](#).

Hint: Compute the exact derivative explicitly.

Hint: Your plot should be clean and readable. That is, you should include axis labels and a legend. You may need to change the line style for your lines to distinguish them.

Hint: Your plot will contain four lines: Analytical derivative, $f'_{FD}(x)$ for $\epsilon = 1 \times 10^{-1}$, $f'_{FD}(x)$ for $\epsilon = 1 \times 10^{-7}$ and $f'_{FD}(x)$ for $\epsilon = 1 \times 10^{-15}$

c) **5 points**

Answer the following questions using two print statements to display your answers. These can be placed at the very end of [P3.py](#). Each print statement should start with the string "Answer to Q-k:" where k is either i or ii to reference the questions below.

- i) Which value of ϵ most closely approximates the true derivative? What happens for values of ϵ that are too small? What happens for values of ϵ that are too large?

ii) How does automatic differentiation address these problems?

Problem 4: Evaluation Trace of Simple Neural Network (25 points)

Deliverables:

1. [P4_graph.png](#)
2. [P4_table.png](#) (or [P4_table.md](#) or [P4_table.ipynb](#))

Note: when you submit images of handwritten solutions, please make sure they are legible. You may lose points if it is not possible to read your submitted work. Thank you.

Artificial neural networks take as input the values of an input layer of neurons and combine these inputs in a series of hidden layers to compute an output. Such a network can be written as $y = f(x)$ with x being the input and y the output. A small network with a single hidden layer is shown in figure 1.

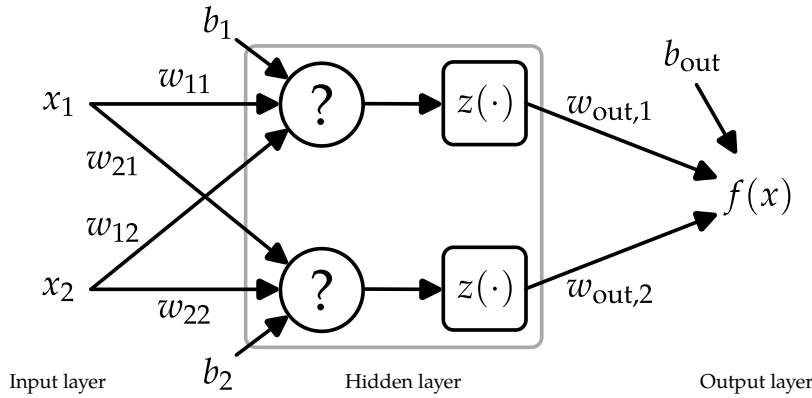


FIGURE 1: Simple neural network with one hidden layer.

This network can be expressed in matrix notation as

$$f(x) = w_{out}^T z(Wx + b) + b_{out}, \quad (2)$$

where $x = [x_1, x_2]^T$ is the input vector, $b = [b_1, b_2]^T$ is a bias vector and

$$W = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} \quad (3)$$

is a weight matrix. Note that there are two output weights applied to the two units in the hidden layer shown in figure 1. These output weights are given by the column vector

$$w_{out} = \begin{bmatrix} w_{out,1} \\ w_{out,2} \end{bmatrix}. \quad (4)$$

The (often) non-linear function $z(q)$ is called activation function and acts component-wise on some input q . Note that because the network only has one output, the function $f(x)$ is a scalar function (its return value is a scalar). Therefore, the value b_{out} and the inner product $w_{out}^T z(Wx + b)$ in equation 2 are both numbers in \mathbb{R} .

In the following, we will completely ignore the bias terms, $b = 0$ and $b_{\text{out}} = 0$. The mathematical form is therefore

$$f(x) = w_{\text{out}}^T z(Wx). \quad (5)$$

Note that in practical applications the biases play a key role. However, we have elected to neglect them in this problem such that the computation simplifies.

a) **11 points**

Similar to what we did in the lecture, draw the computational forward graph. Note that you need to figure out the elementary operations hidden by question marks in figure 1. You may treat $z(q)$ as a single elementary operation. Indicate the individual nodes in the computational graph using the notation v_j for node j with $j = -1, 0, 1, 2, \dots$, where v_{-1} and v_0 correspond to independent coordinates x_1 and x_2 , respectively. Submit your solution in the file [P4_graph.png](#). This image can be a picture taken of your graph drawn on a piece of paper or it can be something that you draw electronically.

Hint: The elementary operations in the hidden layer that are masked out with the question mark are additions or multiplications. Your computational graph must indicated these operations.

b) **14 points**

Use the computational graph of task 4a to write out the forward primal and forward tangent traces in analytical form, the same way we did in the lecture. This table can, once again, be done on a piece of paper (your writing must be legible) or in a markdown table. Submit your traces in a table saved in the file [P4_table.png](#). Alternatively you can submit a Markdown table [P4_table.md](#) or a Markdown table in a Jupyter notebook [P4_table.ipynb](#). The table should include the following columns:

Primal trace	Tangent trace
$v_{-1} = x_1$	$D_p v_{-1} = p_1$
$v_0 = x_2$	$D_p v_0 = p_2$
$v_1 = w_{11} v_{-1}$	$D_p v_1 = w_{11} D_p v_{-1} = w_{11} p_1$
$v_2 = \dots$	$D_p v_2 = \dots$

The seed vector $p = [p_1, p_2]^T$ is a *parameter* which you should keep general in your trace table (you will choose particular values once you have found the general expression for $D_p f(x)$). The values on the right side of the equal signs in your table should be expressed in terms of v_j , p_1 , p_2 , $w_{\text{out},1}$, $w_{\text{out},2}$, w_{11} , w_{12} , w_{21} , w_{22} , $z(\cdot)$, and $z'(\cdot)$, where $z'(\cdot)$ denotes the derivative of $z(\cdot)$.

When you are done, you should provide analytical relations for

1. $f(x)$
2. $\partial f / \partial x_1$
3. $\partial f / \partial x_2$

that are expressed in terms of $w_{\text{out},1}$, $w_{\text{out},2}$, w_{11} , w_{12} , w_{21} , w_{22} , $z(\cdot)$, $z'(\cdot)$, x_1 and x_2 only.

Hint: To evaluate the derivative $\partial f / \partial x_1$ you must choose a “direction” (seed vector) $p = [1, 0]^T$. Similarly for $\partial f / \partial x_2$ you require $p = [0, 1]^T$.

Problem 5: Forward Mode AD with Dual Numbers (25 points)

Deliverables:

1. [P5.py](#)

Important: this problem will be fully *auto-graded*. You are expected to work with the skeleton code provided in [code/p5/P5.py](#). Do not change existing code in that skeleton code. Only add your solution code in appropriate places or update values in existing variables. Otherwise you will risk that the auto-grader fails and you lose points. Read carefully the problem statement and ensure your implementation follows the specifications given.

The auto-grader works by running a set of tests on your implementation. It is therefore important that you are throwing exceptions at places in your code where values do not correspond to what is expected.

In this task you will implement a simple dual number class in Python with support for addition and multiplication operations. Your implementation must further support addition and multiplication of a *scalar number* with a dual number object obtained from your implementation. Your dual number implementation will support basic automatic differentiation for functions involving addition and multiplication operators only. You will consider the following second degree polynomial for this problem

$$f(x) = a_2x^2 + a_1x + a_0, \quad (6)$$

where $f(x): \mathbb{R} \mapsto \mathbb{R}$ with $a_0, a_1, a_2 \in \mathbb{R}$ with the first derivative given by

$$\frac{df}{dx} = 2a_2x + a_1. \quad (7)$$

An example application of your implementation could be to evaluate the the function $f(x)$ and its derivative at the point $x = x_1 = 1/2$ for coefficients $a_0 = 2$, $a_1 = 5/2$ and $a_2 = 3$. The exact solution is $f(x_1) = 4$ and $df/dx|_{x=x_1} = 11/2$. A possible forward mode AD code could look like this:

```
>>> from P5 import DualNumber
>>> x1 = 0.5
>>> z = DualNumber(x1) # initialize real part only
>>> a0 = 2.0; a1 = 2.5; a2 = 3.0
>>> f = a2 * z * z + a1 * z + a0
>>> print(f.real); print(f.dual)
4.0
5.5
```

Note that your code must support the following statement

```
>>> f = a0 + a1 * z + z * a2 * z
```

and other combinations as well. You must raise a `TypeError` if any operand has a type different than `int`, `float` or `DualNumber`. The following special methods are required:

`__init__`: initialize the real and dual parts of a dual number. The first argument must be the real part and *is required*. The second argument initializes the dual part and *is optional*. The real part must be stored in an attribute called `real` and the dual part must be stored in an attribute called `dual`. If the dual part is omitted, it must be initialized such that the value of the derivative df/dx would result in the dual part if the dual number is used with a sequence of addition and multiplication operators (see example code above).

`__add__`: support dual number addition where the other operand may possibly be of `int` or `float` type.

`__mul__`: support dual number multiplication where the other operand may possibly be of `int` or `float` type.

`__radd__`: support reflected (swapped) addition.

`__rmul__`: support reflected (swapped) multiplication.

Hint: Have a look at the documentation of `__add__`, `__mul__`, `__radd__` and `__rmul__`. See <https://docs.python.org/3/reference/datamodel.html>.

Problem 6: Continuous Integration using GitHub Actions (10 points)

Deliverables:

1. `P6.yml`
2. `P6.log`
3. `test_P6.py`

In this problem you will setup your first continuous integration (CI) jobs. CI is configured in simple configuration files where usually YAML² format is used. There are different CI providers like CircleCI, AppVeyor, Azure pipelines, GitLab CI or GitHub actions. We will use GitHub actions for this class because Harvard provides self-hosted runners on <https://code.harvard.edu>. CI providers usually charge a fee for their service as well.

a) 4 points

Install a GitHub workflow in your private class repository and name the file `P6.yml`. You must save this file in `.github/workflows/P6.yml`, where the `.github` directory must be in the root of your repository (where you also have the `.git` repository). You can create a symbolic link to `.github/workflows/P6.yml` in your `homework/hw4` directory if you prefer to work from there (Git allows for *relative* symbolic links within the repository). Make sure you place a copy of the file inside your submission directory when you are done with this task. Note that you have already done something similar for your 1B milestone in the project.

The `P6.yml` workflow should have the following basic configuration:

1. It should be named “HW4 Continuous Integration”.
2. It must be triggered when you push to the branch `hw4`.

You can commit these initial configuration and push to your remote on <https://code.harvard.edu>. You can now visit your repository in the browser and click on the “Actions” tab in the top bar. If your configuration is correct, you should see a workflow with the name of your commit message and this workflow should fail because you have not defined a *job* to run yet.

The next thing you must add in your `P6.yml` workflow is a “`jobs:`” key that configures what should be run when the CI container runs on the remote server.³ Create a job with the following properties:

1. The job ID is `my_first_ci_job`.
2. It should run on the latest Ubuntu Docker container.
3. It should define one step which runs the `date` command followed by the `hostname` command. These are shell commands, you can check what they do on your local system. See for example `man date` and `man hostname`.⁴

²<https://en.wikipedia.org/wiki/YAML>

³<https://docs.github.com/en/actions/using-workflows/workflow-syntax-for-github-actions#jobs>

⁴https://docs.github.com/en/actions/using-workflows/workflow-syntax-for-github-actions#jobsjob_idsteps

Commit these additions and push again to your remote repository. Visit again the “Actions” tab in your repository online and verify the CI job ran successfully (you should see a green check mark for the commit you have just pushed). Click on the successful workflow and then click on the name you gave the job contained within the P6.yml file (inside the gray box). Investigate what has been run when your job was executing. In the top right of the job screen you can see a gear icon. Click on the icon and then click on “View raw logs”. This log tells you what the CI container was running. Your command and output should be at the very bottom. Save this log in a text file called [P6.log](#).

b) *6 points*

Your previous first job was very simple and did not require access to files in your Git repository. In this part you are implementing a few test cases to test a simple implementation of complex numbers.⁵ The untested implementation is provided in [code/p6/P6.py](#). You are required to implement the test cases in the provided skeleton [code/p6/test_P6.py](#) and use pytest to run your tests.⁶ Note that it is important that your tests (which are just Python modules) start with “test_” such that pytest can find them automatically.

A test module will always import names that you need to test and other Python modules or packages that may be required. Your test_P6.py module should start with the following:

```
import pytest # you may need functionality from pytest
from P6 import Complex # test target is the Complex class in P6.py
```

In such test modules you can either define functions that perform tests or combine multiple tests in a class. It is again important that functions (or member functions) that perform tests start with “test” and classes that combine multiple test member functions start with “Test”. This way pytest can automatically infer your tests and minimize your configuration efforts.

To test your code using pytest, you can simply use the assert keyword.⁷ For example, if you have a function func(x) and you want to test it, you could write code like this:

```
def func(x):
    return x + 1

def test_answer():
    assert func(3) == 5
```

This test would of course fail. You can find the output of pytest for this example here <https://docs.pytest.org/en/7.1.x/getting-started.html#create-your-first-test>. Please implement the test cases outlined for you in the [test_P6.py](#) skeleton code. You can always run your tests locally by just executing

⁵Note that this is exactly how autograders work. They are just a collection of tests applied on an untested code.

⁶<https://docs.pytest.org/en/7.1.x/contents.html>

⁷https://docs.python.org/3/reference/simple_stmts.html#the-assert-statement

```
$ pytest
```

in your terminal. (Provided you followed the naming convention explained above.) Once your tests work, you want to integrate them in your CI configuration. Add the following lines after your “my_first_ci_job” in your P6.yml workflow from task 6a:

```
test_complex:
  name: Test Python implementation of complex numbers
  runs-on: ubuntu-latest
  steps:
    # Checkout the head of branch `hw4` (what you just pushed)
    # See: https://github.com/actions/checkout
    - uses: actions/checkout@v3
    # Setup Python environment in the container
    # See: https://github.com/actions/setup-python
    - uses: actions/setup-python@v3
    with:
      python-version: '3.10' # let's use a recent version
    - name: Install Python dependencies
      run: TODO you definitely depend on pytest and maybe others
    - name: Run complex number test suite
      run: TODO execute the tests (this will depend on a path. It
        should at least contain homework/hw4)
```

and replace the TODO items with actual commands that perform the required tasks. These commands are shell commands like python, cd or pytest. Remember that the environment where these commands are run is simply a Docker container running Ubuntu Linux in this case.

Complete the test_complex job such that your test suite is run on every push to the hw4 branch in your private class repository. If your tests fail, you would usually get an email notification. You can try to change something in the Complex class inside P6.py that would not pass your tests. Commit and push the breaking changes and check that your CI tests fail. You can inspect the logs of your GitHub actions in the “Actions” tab on the website of your private class repository.