Homework 2 Bonus
Autograd CNN

11-785: Introduction to Deep Learning (Fall 2023)

OUT: **September 25, 2023**
DUE: **November 11 2023, 11:59 PM EST**

Start Here

- **Collaboration policy:**
  - You are expected to comply with the University Policy on Academic Integrity and Plagiarism.
  - You are allowed to talk with / work with other students on homework assignments
  - You can share ideas but not code, you must submit your own code. All submitted code will be compared against all code submitted this semester and in previous semesters using MOSS.

- **Overview:**
  - **My Grad:** An explanation of the library structure, the local autograder, and how to submit to Autolab. You can get the starter code from Autolab or the course website.
  - **Autograd:** An introduction to Automatic Differentiation and our implementation of Autograd, the various classes, their attributes and methods,
  - **Building Autograd:** The assignment, building the autograd engine, adding backward operations, and using this toolkit to build an CNN!
  - **Appendix:** This contains information on some theory that will be helpful in understanding the homework.

- **Directions:**
  - You are required to do this assignment in the Python (version 3) programming language. Do not use any auto-differentiation toolboxes (PyTorch, TensorFlow, Keras, etc) - you are only permitted and recommended to vectorize your computation using the Numpy library.
  - We recommend that you look through all of the problems before attempting the first problem. However we do recommend you complete the problems in order, as the difficulty increases, and questions often rely on the completion of previous questions.
  - If you haven't done so, use pdb to debug your code effectively (One much simple way, just use print()! Print the shape or anything else that can help you find the bug!)and please PLEASE Google your error messages before posting on Piazza.
  - Complete Autograd bonus 1 before attempting Autograd bonus 2.

1 My Grad Structure

In HW2P1, your implementation of MyTorch worked at the convolutional neural network - thus, stacking several CNN layers followed by activations allowed you to build your very own CNN.

In previous Autograd bonus, we built an alternative implementation of MyTorch based on a popular Automatic Differentiation framework called Autograd that works at the granularity of a single operation. As you will discover, this alternate implementation more closely resembles the internal working of popular Deep Learning frameworks such as PyTorch and TensorFlow (version 2.0 onwards), and offers more flexibility in building arbitrary network architectures. In this bonus, we will keep implementing this my_grad library to allow them to compute CNN and build our own CNN!

For Homework 2 Autograd Bonus, MyTorch will have the following structure:

handout

- mytorch/
  - autograd_engine.py (copy from autograd bonus 1)
  - utils.py
  - sandbox.py
  - nn/
    - functional_hw1.py (Your solution to HW1 Autograd)
    - functional_hw2.py (functionals that you will be implementing in HW2 Autograd)
    - modules/
      - conv.py
      - resampling.py
      - activation (Your solution to HW1 Autograd)
      - loss (Your solution to HW1 Autograd)
- hw2_bonus/
  - data
  - cnn.py
  - cnn_runner.py
- autograder/
  - runner_hw2.py
  - helpers.py
  - test_conv.py
  - test_basic_functional.py
  - test_conv_functional.py

- To make it simpler for you to work on Homework2 Autograd on top of your solution to Homework1 Autograd, you can use

  ```
  tar -xvf hw2_autograd.sh
  ```

in your directory of Homework1 Autograd to add files for this homework.

- **Install** Python3, NumPy and PyTorch in order to run the local autograder on your machine:

```
pip install numpy
pip install torch
```

- **Hand-in** your code by running the following command from the top level directory, then **SUBMIT** the created *handin.tar* file to autolab:

```
sh create_tarball.sh
```

- **Autograde** your code by running the following command from the top level directory:

```
python3 autograder/runner_hw2.py
```

- **DO:**
  - We strongly recommend that you understand how the Autograd Engine works (like get 100 points in previous bonus) as they will be essential in this homework.
- **DO NOT:**
  - Import any other external libraries other than numpy, as extra packages that do not exist in autolab will cause submission failures. Also do not add, move, or remove any files or change any file names.

2 Building Autograd

In this section, you need to implement convolutional neural networks using the Autograd Enigne.

Your implementations will be compared with PyTorch.

2.1 Adding Operation Backward Functions[10 points]

Recall that Autograd is defined to work at the operation level and not the layer level. This means that we must define a backward function for every operation that we will use. Note that you have already calculated and implemented these backward functions in HW2P1 - you just need to define them individually and explicitly now.

2.1.1 In nn/functional_hw2.py complete the following [10]

These two functions are almost same as we implement in the HW2P1.

- conv1d_stride1_backward + downsampling1d_backward[5]
- conv2d_stride1_backward + downsampling2d_backward[5]
- flatten_backward[0]

2.2 Building a Network [10 points]

Finally, in hw2 bonus/cnn.py you can put to use the new Autograd module that you have so painstakingly crafted to actually build a Neural Network.

2.2.1 A simple CNN layer[10]

- Conv1d_stride1[0]
- Conv1d[5]
- Conv2d_stride1[0]
- Conv2d[5]
- Flatten[0]

In the file nn/modules/cnn.py complete the CNN class and Flatten class. Your task is to complete the forward() function of this class and it is same as the Conv class you implement in HW2P1. We will use Autograd Engine in backward() function of this class to automatically do back-propagation:

(1) Compute the outputs for the forward computation from the given input and the layer parameters,

(2) Note the parameters that are being passed to the forward method,

(3) Remember the syntax for using the add_operation() function defined in autograd_engine.py,

(4) Also remember to ensure that the input parameters are in the same order as the gradients to update parameters.

Why return to the Layer Level abstraction if Autograd lives at the Operation Level Abstraction?

Defining Layer classes is an effort to regress to good coding practices. Concretely, we achieve the following by doing this:

- We hide the autograd engine object and do not explicitly expose its internals to the user,
- We make our code more modular - defining a Linear Layer class for example, avoids repeatedly making calls to the add operation function of the autograd engine,
- This is how most deep learning frameworks such as PyTorch define layers and we believe this homework will help solidify that syntax in your mind.

2.2.2 Building a CNN[Ungraded]

Finally in hw2_bonus.py stack CNN layers and activations together to build an the CNN class by implementing the following network architecture:

Input → CNN → Tanh → CNN → ReLU → CNN → Sigmoid → Flatten → Output

Note that this section is not graded and is simply provided as proof-of-concept for the Autograd library that you have built.