Homework 2 Autograd

Applying Autograd to Convolutional Networks

11-785: Introduction to Deep Learning (Spring 2024)

OUT: **February 9th**
DUE: **March 9th, 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!

- **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. Alternatively (and more simply) you can use `print()` statements. Print the shape or anything else that can help you find the bug!) and 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 the 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 library to allow you to construct your own CNN! This assignment is shorter than Autograd 1, since by now, you have already implemented the autograd engine. If you have not completed Autograd 1, we $\sim$ strongly $\sim$ suggest you go through it (you will need a working autograd engine anyway).

Combining your work from Homework 1 Autograd with this Homework 2 Autograd assignment, `mytorch` will have the following structure:

`handout`

- `mytorch`
  - `autograd_engine.py` (Copy over your solution to HW1 Autograd)
  - `functional_hw1.py` (Copy over your solution to HW1 Autograd)
  - `functional_hw2.py`
  - `sandbox.py`
  - `utils.py`
  - `nn`
    - `activation.py` (Copy over your solution to HW1 Autograd)
    - `conv.py`
    - `linear.py` (Copy over your solution to HW1 Autograd)
    - `loss.py` (Copy over your solution to HW1 Autograd)
    - `resampling.py`
- `autograder`
  - `helpers.py`
  - `runner_hw2.py`
  - `test_conv.py`
  - `test_basic_functional.py`
  - `test_conv_functional.py`

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

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

  You can also use the environment used on previous homework part 1s to ensure versions align with autolab:

```
conda activate idls24
```

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

  ```
  tar -cvf handin.tar mytorch
  ```

- **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 (read: get 100 points in previous bonus) as they will be essential in this homework.
- **DO NOT:**
  - Import any other external libraries other than `numpy` or `torch` or that are not in the environment from previous homeworks, 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 Primitive Operations for Convolutions [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. Note, however, that in our implementation, we are defining the convolution as a single operation. That is, these functions will be very similar to what you implemented in HW2P1.

In `functional_hw2.py` complete the following functions.

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

2.2 Building a Convolutional Layer [10 points]

Now, we can use the primitive operations defined above to create 1D and 2D convolutional layers, similar to what you might use in PyTorch.

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 convolutional 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.

To do so, you will have to complete the files in `nn/conv.py` and `nn/resampling.py` as laid out below.

- In `nn/conv.py`:
  - `Conv1d_stride1` + `Conv1d` [5]
  - `Conv2d_stride1` + `Conv2d` [5]
  - `Flatten` (Not strictly necessary, but you can implement easily).
- In `nn/resampling.py`:
  - `Downsample1d` (This will be necessary for your implementation of the `Conv1d` class.)
  - `Downsample2d` (This will be necessary for your implementation of the `Conv2d` class.)
  - `Upsample1d` (Not strictly necessary, but you can implement easily).
  - `Upsample2d` (Not strictly necessary, but you can implement easily).

Pay attention to where you add operations to the forward passes of these classes. As a hint, you can check where we pass an autograd engine to the constructor function of the class (you may need it there to add an operation).

2.3 Building a CNN [Ungraded]

Now that you have built convolutional layers and activations, try stacking them together to build a 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.