HW2P1 Bonus

Dropout2d, BatchNorm2d and ResNet

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

Out: **9th February 2024, 11:59 PM EST**

Due: **23rd March 2024, 11:59PM EST**

Start Here

Collaboration policy

- You are expected to comply with the University Policy on Academic Integrity and Plagiarism1
- 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

- Dropout 2D
- BatchNorm 2D
- ResNet

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.
- If you haven't done so, use pdb to debug your code effectively.

1 MyTorch

The culmination of all of the Homework Part 1's will be your own custom deep learning library, which we are calling MyTorch. It will act similar to other deep learning libraries like PyTorch or Tensorflow. The files in your homework are structured in such a way that you can easily import and reuse modules of code for your subsequent homework.

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

- mytorch
  - batchnorm2d.py
  - dropout2d.py
  - Conv2d.py
  - activation.py
  - resampling.py
- models
  - resnet.py
- autograder
  - hw2_bonus_autograder
    - runner.py
    - test.py
- create_tarball.sh
- exclude.txt

- **Install** Python3, NumPy and PyTorch in order to run the local autograder on your machine:
  - pip3 install numpy
  - pip3 install torch
- **Autograde** your code by running the following command from the top level directory:
  - python3 autograder/hw2_bonus_autograder/runner.py
- **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
- **DO**
  - We strongly recommend that you review the lectures on CNNs.

- **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 Dropout 2D [5 points]

Dropout2D randomly zeroes out entire channels (a channel is a 2D feature map, e.g., the j-th channel of the i-th sample in the batched input is a 2D array of shape (*Channel Width*, *Channel Height*). Each channel will be zeroed out independently on every forward call with probability p using samples from a Bernoulli distribution. In this bonus, you will implement both forward propagation and backward propagation for Dropout2D. Your implementation should be similar to your implementation of dropout in the HW1 bonus.

Implementation Notes:

- You should use `np.random.binomial` to select the channels and `np.tile` for creating the mask.
- You should scale during training but not during inference.

3 BatchNorm 2D [5 points]

Applies Batch Normalization over a 4D input (a mini-batch of 2D inputs with additional channel dimension) as described in the paper Batch Normalization.

Note that 2D Batch Normalization is done over the channel dimension, computing statistics on (Batch, Height, Width) slices, it's common terminology to call this Spatial Batch Normalization. You can play with PyTorch's BatchNorm2d layer to understand how it works.

Refer to the BatchNorm formulae provided in HW1P1 handout for implementation details.

4 ResNet [5 points]

ResNet, also known as Residual Neural Network, is a popular deep learning model that incorporates skip connections between layers. By incorporating this model we can tackle the problem of vanishing gradients that arise when training extremely deep neural networks.

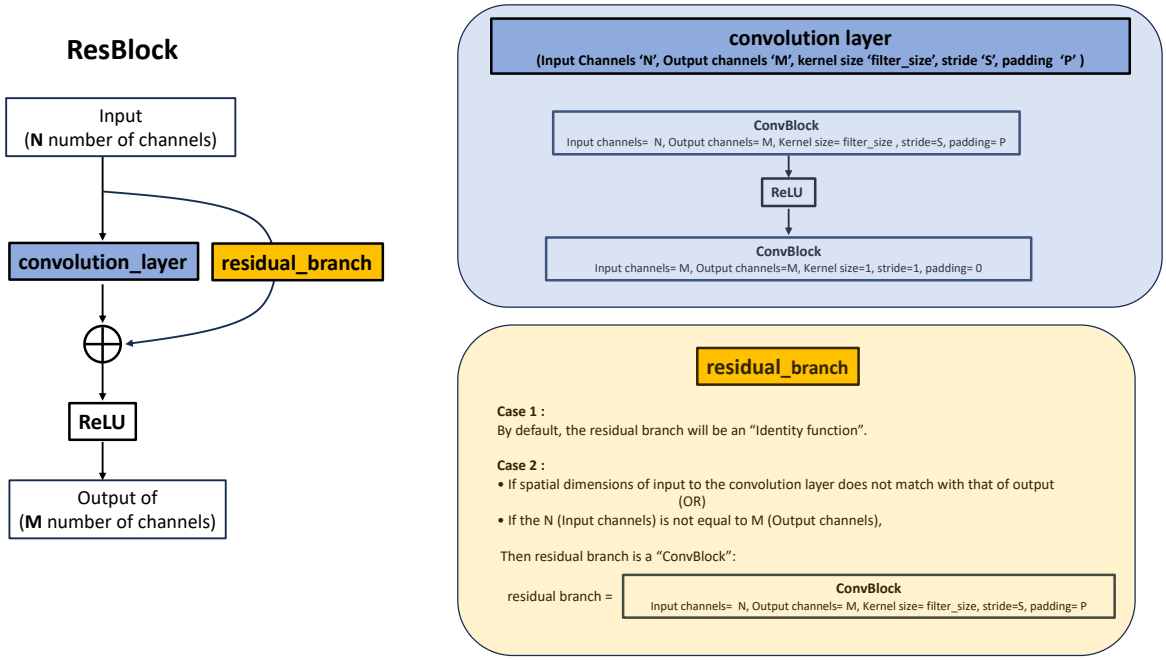

Figure 1: Residual Neural Network

In this section, your task is to implement a ResNet block. Please consider the following class structure.

[Note: A ConvBlock consists of 2D convolution followed by 2D batch normalization.]

```
class ConvBlock(object):

    def __init__(self, in_channels, out_channels, kernel_size, stride, padding):
       self.layers = []                                    #TODO

    def forward(self, A):
                                                           #TODO
       return NotImplemented

    def backward(self, grad):
                                                           #TODO
       return NotImplemented


class ResBlock(object):

    def __init__(self, in_channels, out_channels, filter_size, stride=3, padding=1):

       self.convolution_layers =  []                       #TODO
```

```
    self.final_activation = None                #TODO

    if stride != 1 or in_channels != out_channels or filter_size!=1 or padding!=0:

        self.residual_connection = None         #TODO

    else:
        self.residual_connection = None         #TODO


def forward(self, A):
                                                #TODO
    return NotImplemented

def backward(self, grad):
                                                #TODO
    return NotImplementedError
```