Project #2: Convolutional Neural Networks

Amir Sadovnik
COSC 525: Deep Learning (Spring 2021)

---

1 Overview

In this project you will be adding a few layer types to your artificial neural network library. More specifically, you are required to add the following layers:

1. `ConvolutionalLayer`
2. `MaxPoolingLayer`
3. `FlattenLayer`

In addition you will get your first experience using tensorflow with the keras API. (tf.keras) As in the previous project you are required to code this in the manner described below. Make sure to use your own solution and your own code.

2 Problem Description

Your are tasked with writing the following functions:

1. Change your `NeuralNetwork` class.
   (a) Instead of sending all the information about the layers of the network during initialization, simply initialize the network with the input size, loss function and learning rate.
   (b) Add a method to the class called addLayer. This method should accept all the details needed to initialize the layer (this should include allowing for setting weights. Note that the input size should not be set as a parameter, but simply set to the current final layer.
2. Write a `ConvolutionalLayer` class:

(a) In this project we will restrict the layer to 2d convolutions.

(b) Should be initialized with the number of kernels in the layer, the size of the kernel (assume it is square), the activation function for all the neurons in the layer, the dimension of the inputs, the learning rate, and possibly a vector of weights (if not set to random). For this project we will assume the stride is always 1 and padding is set to valid (if you would like to add a stride parameter or padding it is up to you).

(c) Each neuron (as opposed to each Kernel) should be represented as a Neuron object. Since all neurons with the same kernel share the same weights, make sure to initialize them with the same values.

(d) Should have a `calculate` method which given an input calculates the output of all the neurons in the layer.

(e) Should have a `calculatewdeltas` which given the $\sum w \times \delta$ from the next layer, goes through all the neurons in the layer to calculate their partial derivative (using the `calcpartialderivative` method). Note that since there are shared weights you will need to add the returned $\sum w \times \delta$ in the correct way. In addition, for each filter you will need to add all the partial derivatives fom the different locations to calculate the final update. Then simply updates all the weights (using the `updateweights` method) and returns its own $\sum w \times \delta$.

3. Write a `MaxPoolingLayer` class:

(a) In this project we will restrict the layer to 2d max pooling.

(b) Should be initialized with the size of the kernel (assume it is square), and the dimension of the inputs. For this project we will assume the stride is always the same as the filter size and no padding is needed (if you would like to add a stride parameter, that is up to you).

(c) Should have a `calculate` method which given an input calculates the output of the layer (No need for actual neurons). In order to perform back propagation you will need to store the locations of the max.

(d) Should have a `calculatewdeltas` which given the $\sum w \times \delta$ from the next layer, returns a new $\sum w \times \delta$ which is the size of the input and the values are set in the correct locations (the max).

4. Write a `FlattenLayer` class.

(a) This class only needs to be initialized with the input size.

(b) Should have a `calculate` method which given an input simply resizes it to create the output of the layer (No need for actual neurons).

(c) Should have a `calculatewdeltas` which given the $\sum w \times \delta$ from the next layer, simply resizes it to the size of the input.

5. Your `main` method will compare three setups with results obtained from using keras (in order to verify correctness). For each run one step of back propagation and compare the weights you get using your method with the results from tensorflow:

(a) If given `example1`, run a network with a 5x5 input, one 3x3 convolution layer with a single kernel, a flatten layer, and single neuron for the output.

(b) If given `example2`, run a network with a 7x7 input, one 3x3 convolution layer with two kernels, another 3x3 convolution layer with a single kernel, a flatten layer, and single neuron for the output.

(c) If given `example3`, run a network with a 8x8 input, one 3x3 convolution layer with two kernels, a 2x2 max pooling layer, a flatten layer, and single neuron for the output.

3 Additional Information

You must do so under the following constraints:

1. You can use the same activations and loss functions as the previous project.
2. You must use Python3 with only the numpy, sys, matplotlib libraries allowed.
3. Make sure to comment your code.
4. In order to compare the results with tensorflow/keras you will need to have tensorflow 2.0 installed. Let me know soon if you have any problems.
5. I have provided code for `example2` in Tensorflow to show you how to produce the numbers you need to compare to. You can use that as your base for the other two examples.

4 Report

You should submit a short PDF report with the following (if you do not have anything to add for a section, simply put the section title and then state there is nothing to add):

1. A short introduction to the problem.
2. Assumptions/choices you have made.
3. Problems/Issues you were not able to solve.
4. How to run you code (if there is any difference from what is stated)
5. Show screen shots of your weight comparison for the three examples.

5 Submission

You are required to submit one zip file with the code and your report.