CS 615 - Deep Learning

Assignment 2 - Objective Functions and Gradients
Winter 2022

Introduction

In this assignment we'll implement our output/objective modules and add computing the gradients to each of our modules.

Allowable Libraries/Functions

Recall that you **cannot** use any ML functions to do the training or evaluation for you. Using basic statistical and linear algebra function like *mean*, *std*, *cov* etc.. is fine, but using ones like *train* are not. Using any ML-related functions, may result in a **zero** for the programming component. In general, use the "spirit of the assignment" (where we're implementing things from scratch) as your guide, but if you want clarification on if can use a particular function, DM the professor on slack.

Grading

Do not modify the public interfaces of any code skeleton given to you. Class and variable names should be exactly the same as the skeleton code provided, and no default parameters should be added or removed.

Theory	18pts
Implementation of non-input, non-objective gradient methods	20pts
Implementation of objective layers	40pts
Tests on non-input, non-objective gradient methods	10pts
Tests on objective layers' loss computations	4pts
Tests on objective layers' gradient computations	8pts
TOTAL	100pts

Table 1: Grading Rubric

1 Theory

1. (8 points) Given $h = \begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix}$ as an input, compute the gradients of the output with respect to this input for:

   (a) A ReLu layer

   (b) A Softmax layer

   (c) A Sigmoid Layer

   (d) A Tanh Layer

2. (2 points) Given $h = \begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix}$ as an input, compute the gradient of the output a fully connected layer with regards to this input if the fully connected layer has weights of $W = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \\ 7 & 8 \end{bmatrix}$ as biases $b = \begin{bmatrix} -1 & 2 \end{bmatrix}$.

3. (2 points) Given a target value of $y = 0$ and an estimated value of $\hat{y} = 0.2$ compute the loss for:

   (a) A squared error objective function

   (b) A log loss (negative log likelihood) objective function)

4. (1 point) Given a target *distribution* of $y = [1, 0, 0]$ and an estimated distribution of $\hat{y} = [0.2, 0.2, 0.6]$ compute the cross entropy loss.

5. (4 points) Given a target value of $y = 0$ and an estimated value of $\hat{y} = 0.2$ compute the gradient of the following objective functions with regards to their input, $\hat{y}$:

   (a) A squared error objective function

   (b) A log loss (negative log likelihood) objective function)

6. (1 point) Given a target *distribution* of $y = [1, 0, 0]$ and an estimated distribution of $\hat{y} = [0.2, 0.2, 0.6]$ compute the gradient of the cross entropy loss function, with regard to the input distribution $\hat{y}$.

2 Adding Gradient Methods

To each of your non-input modules from HW1 (*FullyConnectedLayer*, *ReLuLayer*, *SoftmaxLayer*, *TanhLayer* and *SigmoidLayer*) implement the *gradient* method such that it computes and returns the gradient (as a single float value) of the most recent output of the layer with respect to its most recent input (both of which should have been stored in the parent class, and updated in the *forward* method).

3 Objective Layers

Next, let's implement a module for each of our objective functions. These modules should implement (at least) two methods:

- *eval* - This method takes two explicit parameters, the target value and the incoming/estimated value, and computes and returns the loss (as a single float value) according to the module's objective function.
- *gradient* - This method takes the same two explicit parameters as the *eval* method and computes and returns the gradient of the objective function using those parameters.

Implement these for the following objective functions:

- Least Squares as *LeastSquares*
- Log Loss (negative log likelihood) as *LogLoss*
- Cross Entropy as *CrossEntropy*

Your public interface is:

```
class XXX():
  def eval(self,y, yhat):
    #TODO

  def gradient(self,y, yhat):
    #TODO
```

4 Testing the gradient methods

For each non-input, non-objective layer (*FullyConnectedLayer* (w/ two outputs), *ReLuLayer*, *SoftmaxLayer*, *TanhLayer*, and *SigmoidLayer*):

1. Instantiate the layer (for reproducibility, seed your random number generator to zero prior to running your tests).
2. Forward propagate through the layer using the input provided below.
3. Run your gradient method to get the gradient of the output of the layer with respect to its input.

Here's the input (single observation) to each of these layers:

$$h = \begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix}$$

In your report provide the gradient returned by each layer.

5 Testing the Objective Layers

Finally we'll test the objective layers. For each objective module you'll:

- Instantiate the layer/module
- Evaluate the objective function using the provide estimate and target value(s).
- Compute (and return) the gradient of the objective function given the estimated and target value(s).

For this you'll use the following target (y) and estimated ($\hat{y}$) values for the *least squares* and *log loss* objective functions:

$$y = 0$$

$$\hat{y} = 0.2$$

and the following for the cross-entropy objective function:

$$y = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$$

$$\hat{y} = \begin{bmatrix} 0.2 & 0.2 & 0.6 \end{bmatrix}$$

In your report provide the evaluation of the objective function and the gradient returned by each of these output layers.

Submission

For your submission, upload to Blackboard a single zip file containing:

1. PDF Writeup
2. Source Code
3. readme.txt file

The readme.txt file should contain information on how to run your code to reproduce results for each part of the assignment.

The PDF document should contain the following:

1. Part 1: Your solutions to the theory question
2. Parts 2-3: Nothing
3. Part 4: The gradient of the output of each layer with respect to its input, where the provided X is the input.
4. Part 5: The loss of each objective layer using the provided y and $\hat{y}$ as well as the gradient of the objective functions, with regards to their input ($\hat{y}$).