CS 615 - Deep Learning

Assignment 2 - Objective Functions and Gradients
Winter 2023

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	20pts
Testsing fully-connected and activation layers' gradient methods	40pts
Testing objective layers' loss computations and gradients	40pts
TOTAL	100pts

Table 1: Grading Rubric

1 Theory

1. (10 points) Given $H = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$ as an input, compute the gradients of the output with respect to this input for the following activation layers. Show your answer in **tensor form** by having a Jacobian matrix for each observation.

   (a) A ReLu layer

   For the Relu activation function, the gradient can be calculated as
   $\frac{\partial g_j(z)}{\partial z_j} = 0, if z_j < 0$
   $\frac{\partial g_j(z)}{\partial z_j} = 1, if z_j >= 0$
   The output of the gradient only depends on the negative and positive values, it will give an array of shape NxKxK where all the elements are zero which are not on the diagonal. On the diagonal the gradient will return 1 if the output is more than zero and one if vice-versa. Using the above formula we get the gradient of reLu function as
   $[\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
   $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}]$

   (b) A Softmax layer

   The output of the Softmax function can be given by the formula
   $\frac{\partial g_j(z)}{\partial z_j} = g_j(z)(1 - g_j(z))$ if, i == j
   $\frac{\partial g_j(z)}{\partial z_j} = -g_i(z).g_j(z)$, if, i !=j
   where g(z) is my softmax function. Using this we get the tensor value as
   $[\begin{bmatrix} 0.08192507 & -0.02203304 & -0.05989202 \\ -0.02203304 & 0.18483645 & -0.1628034 \\ -0.05989202 & -0.1628034 & 0.22269543 \end{bmatrix}$
   $\begin{bmatrix} 0.08192507 & -0.02203304 & -0.05989202 \\ -0.02203304 & 0.18483645 & -0.1628034 \\ -0.05989202 & -0.1628034 & 0.22269543 \end{bmatrix}]$

   (c) A Sigmoid Layer

   Sigmoid function gradient will also work in the similar fashion giving us all the values as zero in the tensor of NxKxK except for the diagonal elements. On the diagonal elements of the tensor, the formula to be applied is
   $\frac{\partial g_j(z)}{\partial z_j} = g_j(z)(1 - g_j(z))$
   The tensor output looks like:
   $[\begin{bmatrix} 0.19661193 & 0 & 0 \\ 0 & 0.10499359 & 0 \\ 0 & 0 & 0.04517666 \end{bmatrix}$
   $\begin{bmatrix} 0.01766271 & 0 & 0 \\ 0 & 0.00664806 & 0 \\ 0 & 0 & 0.00246651 \end{bmatrix}]$

(d) A Tanh Layer For this activation function the tensor will be of the form NxKxK where all the non diagonal elements will be zero and all the diagonal elements will give a value derived through this formula
$\frac{\partial g_j(z)}{\partial z_j} = 1 - g_j(z)^2$
This will give the tensor as
$$[\begin{bmatrix} 4.19974342e-01 & 0 & 0 \\ 0 & 7.06508249e-02 & 0 \\ 0 & 0 & 9.86603717e-03 \end{bmatrix}$$
$$\begin{bmatrix} 1.34095068e-03 & 0 & 0 \\ 0 & 1.81583231e-04 & 0 \\ 0 & 0 & 2.45765474e-05 \end{bmatrix}]$$

(e) A Linear Layer
Since linear layer is given by the function $g(z) = z$, its derivative is simply 1 or an identity matrix. Hence the output of the following gradient will be a NxKxK identity tensor.
$$[\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}]$$

2. (2 points) Given $H = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$ 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 $W = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$ as biases $b = \begin{bmatrix} -1 & 2 \end{bmatrix}$.

When computing the output of the fully connected layer, we will use the expression
$g(z) = x.W + b$ where W is my weight matrix and b is my bias matrix.

Stating that when I differentiate my output function with respect to the input I will get nothing but the tensor of W transpose.

my gradient will look like
$$[\begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$$
$$\begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}]$$

3. (2 points) Given target values of $Y = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ and estimated values of $\hat{Y} = \begin{bmatrix} 0.2 \\ 0.3 \end{bmatrix}$ compute the loss for:

(a) A squared error objective function
squared error objective function can be computed by the formula
$\sum_{i=1}^{D}(y_i - yhat_i)^2$

Here my D is 2 since i have 2 values in my y and yhat matrices.
This formula will give result as : 0.26

(b) A log loss (negative log likelihood) objective function)
Log loss or negative log likelihood can be calculated by the objective function
$-\sum_{i=1}^{D}(y_i \log(yhat_i) + (1-y_i)\log(1-yhat_i))$
This formula will give value as: 0.71

4. (1 point) Given target *distributions* of $Y = \begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\end{bmatrix}$ and estimated distributions of $\hat{Y} = \begin{bmatrix}0.2 & 0.2 & 0.6\\ 0.2 & 0.7 & 0.1\end{bmatrix}$ compute the cross entropy loss.

For the cross entropy loss, the formula is
$\sum_{i=1}^{D} y_i.log_2 Yhat_i$
using this formula the loss computed is 0.98

5. (4 points) Given target values of $Y = \begin{bmatrix}0\\ 1\end{bmatrix}$ and estimated values of $\hat{Y} = \begin{bmatrix}0.2\\ 0.3\end{bmatrix}$ compute the gradient of the following objective functions with regards to their input, $\hat{Y}$:

(a) A squared error objective function
Since the gradient of the squared error objective function can be given by the formula
$gradient = 2(y - yhat)$
The output computed will be $\begin{bmatrix}0.4\\ -1.4\end{bmatrix}$

(b) A log loss (negative log likelihood) objective function)
Since the gradient of the log loss objective function can be given by the formula
$gradient = -\frac{y}{yhat} + \frac{1-y}{1-yhat}$
Putting the values of y and yhat matrices in the formula to calculate the gradient we get
$\begin{bmatrix}1.25\\ -3.33\end{bmatrix}$

6. (1 point) Given target *distributions* of $Y = \begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\end{bmatrix}$ and estimated distributions of $\hat{Y} = \begin{bmatrix}0.2 & 0.2 & 0.6\\ 0.2 & 0.7 & 0.1\end{bmatrix}$ compute the gradient of the cross entropy loss function, with regard to the input distributions $\hat{Y}$.

The formula for the gradient of the cross entropy objective function can be written as
$gradient = -\frac{y}{yhat}$
The gradient will be
$\begin{bmatrix}-4.9 & 0 & 0\\ 0 & -1.4 & 0\end{bmatrix}$

2 Update Your Codebase

In this assignment you'll add gradient methods to your existing fully-connected layer and activation functions, and implement your objective functions.

Adding Gradient Methods

Implement *gradient* methods for your fully connected layer, and all of your activation layers. When applicable, you may decide whether a class' gradient method returns a matrix or a tensor. The prototype of these methods should be:

```
#Input: None
#Output:  Either an N by D matrix or an N by (D by D) tensor
def gradient(self):
    #TODO
```

Adding Objective Layers

Now 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 values and the incoming/estimated values, 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:

- Squared Error as *SquaredError*
- Log Loss (negative log likelihood) as *LogLoss*
- Cross Entropy as *CrossEntropy*

Your public interface is:

```
class XXX():
  #Input: Y is an N by K matrix of target values.
  #Input: Yhat is an N by K matrix of estimated values.
  #Output: A single floating point value.
  def eval(self,Y, Yhat):
    #TODO

  #Input: Y is an N by K matrix of target values.
  #Input: Yhat is an N by K matrix of estimated values.
  #Output:  An N by K matrix.
  def gradient(self,Y, Yhat):
    #TODO
```

3 Testing the gradient methods

Let's test our gradient methods! We'll use the same data from the theory component.

Write a script that:

1. Instantiates the fully-connected layer with three inputs and two outputs.
2. Instantiates each activation layer.
3. Sets the weights and biases of your fully-connected layer to $W = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$ and $b = \begin{bmatrix} -1 & 2 \end{bmatrix}$, respectively.
4. Passes the data $H = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$ through the *forward* method of each aforementioned layer.
5. Calls each aforementioned layer's *gradient* method, printing its output.

4 Testing the Objective Layers

Finally we'll test the objective layers.

Write a script that:

- Instantiates each objective function.
- Evaluates the objective functions using the provide estimate and target value(s), printing their output.
- Runs the objective functions' *gradient* method given the estimated and target value(s).

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

$$Y = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$\hat{Y} = \begin{bmatrix} 0.2 \\ 0.3 \end{bmatrix}$$

and the following for the cross-entropy objective function:

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

$$\hat{Y} = \begin{bmatrix} 0.2 & 0.2 & 0.6 \\ 0.2 & 0.7 & 0.1 \end{bmatrix}$$

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. Part 2: Nothing
3. Part 3: The gradient of the output of each layer with respect to its input, where the provided H is the input.
4. Part 4: 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 ($\hat{Y}$).