HOMEWORK 2 - Back Propagation

INDEX

Abbreviation List

N	Sample size/ Dataset Size
σ()	Sigmoid Function
x_i	Input vector
L	Loss Function

Introduction

Backpropagation is an conceptually simple and computationally efficient algorithm that is used to calculate the gradient using partial derivative of loss function w.r.t. the weights (in our case matrix A,B and C) by working backward from output nodes to input nodes. It is considered as an efficient way of calculating the gradients because it avoids redundant calculations thus saving time for model training.

Here, for this assignment I have implemented a simple back propagation algorithm that is purely for learning, therefore parameters like learning rate, epochs and such has not been finely tuned to minimize the loss function to get the lowest possible value.

Problem definition and Algorithm

Task Definition

The main idea behind the assignment is to explore the algorithm of back propagation and have a better understanding of why is it preferred nowadays.

We are provided with each computational layers (ie, the equations) and have to derive the derivatives using back propagation manually and use the gradients to find A, B and C which minimizes the Loss function.

$$(\widehat{A}, \widehat{B}, \widehat{C}) := \arg \min_{(A,B,C)} \sum_{i=1}^{N} \mathcal{L}(x_i; A, B, C)$$

Algorithm Definition

1. Sigmoid Function

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Sigmoid function

where *X* is any variable for which the sigmoid function is calculated. To calculate the sigmoid, I have included both cases for when gamma is '+ve' and '-ve', as if gamma is a large negative value exp() will return a large positive value which results in a *Overflow / math range error for log or exp.*

2. Sigmoid Function Derivative

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

Derivative of the Sigmoid Function

where *X* is any variable for which the derivative of sigmoid function is calculated.

3. Among other mathematical operations, I have included:

    - Matrix - Vector Multiplication
    - Vector - Matrix Multiplication

- Multiplication of 2 Vectors
- Addition of 2 Vectors

4. To minimize the loss function, we decrease the value of each weights (Matrix A, B and C) by some learning rate (lr) towards the global minimum using gradient descent. This can be expressed using the formula,

$$a_{new} = a - lr \times g_a$$
$$b_{new} = b - lr \times g_b$$
$$c_{new} = c - lr \times g_c$$

where a, b and c are the coefficients, lr is the learning rate and g_a , g_b and g_c are the gradients.

5. Equations involved in forward pass

$$\begin{aligned} y &:= Ax \\ u &:= \sigma(y) \\ v &:= Bx \\ z &:= (u+v) \\ w &:= Cz \\ \mathcal{L} &:= \|w\|^2 \end{aligned}$$

6. Derivatives calculated in Backward Pass

① $L = \|w\|^2$

$$\frac{\partial L}{\partial w} = 2w$$

② $w = Cz$

$$\frac{\partial L}{\partial z} = \frac{\partial L}{\partial w} * \frac{\partial w}{\partial z}$$

$$= 2w * C$$

$$\frac{\partial L}{\partial C} = \frac{\partial L}{\partial w} * \frac{\partial w}{\partial C}$$

$$= 2w * z$$

③ $z = (u + v)$

$$\frac{\partial L}{\partial u} = \frac{\partial L}{\partial z} * \frac{\partial z}{\partial u}$$

$$= 2wC * 1$$

$$\frac{\partial L}{\partial v} = \frac{\partial L}{\partial z} * \frac{\partial z}{\partial v}$$

$$= 2wC * 1$$

④ $v = Bx$

$$\frac{\partial L}{\partial B} = \frac{\partial L}{\partial v} * \frac{\partial v}{\partial B}$$

$$= 2wC * x$$

⑤ $u = \sigma(y)$

$$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial u} * \frac{\partial u}{\partial y}$$

$$= 2wC * \sigma(y)(1 - \sigma(y))$$

⑥ $y = Ax$

$$\frac{\partial L}{\partial A} = \frac{\partial L}{\partial y} * \frac{\partial y}{\partial A}$$

$$= 2wC\,\sigma(y)(1 - \sigma(y)) * x$$

Experimental Evaluation

Methodology

1. Import packages

   The first step of this assignment is selecting the required packages and importing them. I have selected libraries for basic computation, so it just helps to make the process smoother but not easier. The libraries are as follows:

   - **Random**: Random is python's built-in module that can be used to generate random numbers. It is mainly used here for generating random matrix of K x K length.
   - **Math**: This module provides access to the mathematical functions defined by the C standard. This exp() of this library is used to calculate the exponent of gamma.
   - **Torch**: Pytorch is used to train and build layers of neural networks. However, here we have only used it for cross checking the value of gradient that is calculated using manual back propagation.

2. Sigmoid Function

   It is used to calculate the sigmoid of gamma and it is called each time when either *du_dy* is to be calculated or when *u* is to be calculated.

3. Backward Pass

   This step involves the calculation of partial derivatives w.r.t. matrix A, B and C.

4. Forward Pass

   Here we are calculating the value of each equation until we are getting the value for final loss function.

5. Gradient Descent

   This function iterates *epoch* times, and during each epoch, it calculates a new value for A, B and C using learning rate and partial derivative. The final gradient value is also cross checked with the value calculated using pytorch to ensure that the value is calculated correctly.

6. Start of the back propagation program

   This is the start of the program for back propagation where we are assigning values for parameters like *N, K, A, B, C, learning_rate, epoch.* Then we are using the randomly calculated vector *x* as the input, to calculate the gradient descent using back propagation. The weights are summed for each x input and then averaged by N. Then this average value of *A, B* and *C* is used to calculate the minimized Loss function.

Observation and Discussion

1. Value for Loss function

```
L =  0.6405446478366267
L =  0.6227243743152856
L =  0.6054711582958173
L =  0.5887633523891213
L =  0.5725803169288799
L =  0.556902363816204
L =  0.5417107039990863
L =  0.5269873983199291
L =  0.5127153114862452
L =  0.4988780689394727
L =  0.485460016414952
L =  0.47244618200257604
L =  0.45982224053267584
L =  0.4475744801254248
L =  0.43568977075459725
L =  0.42415553468797806
L =  0.4129597186772421
L =  0.40209076777970915
L =  0.3915376007032291
L =  0.38128958657351764
L =  0.3713365230306946
L =  0.3616686155686011
L =  0.3522764580367418
L =  0.3431510142304625
L =  0.3342836005002956
L =  0.32566586931628994
L =  0.3172897937276602
L =  0.30914765266224875
L =  0.3012320170141361
L =  0.2935357364712854
L =  0.2860519270383867
L =  0.2787739592130982
L =  0.2716954467766908
```

We can see that the value of loss function is decreasing in each iteration for an input vector x_i.

```
L =  0.0048365386124306535

Gradient calculated manually:
dL_dA =  [[-0.0006713625025565516, 0.0013139067848766854, 0.0018388094918960131], [0.0004449456167331362, -0.000870791952932408
8, -0.0012186713143954573], [-0.0003845423355149779, 0.0007525781999760953, 0.0010532314418636164]]

dL_dB =  [[-0.002701376393878583, 0.005286796267153462, 0.007398859089288121], [0.0017815141111817124, -0.003486556769437744, -
0.004879428096018339], [-0.0015383229336941792, 0.0030106133902554895, 0.004213346442952265]]

dL_dC =  [[-0.014336205554305061, -0.02904568811238444, -0.034179410889230925], [0.027282602131008832, 0.055275571307192434, 0.
06504533328786807], [0.009420241315296222, 0.019085760883592165, 0.0224591017038348]]

Autograd L =  tensor(0.0047, grad_fn=<PowBackward0>)

Gradient calculated using autograd:
 tensor([[-0.0007,  0.0013,  0.0018],
        [ 0.0004, -0.0009, -0.0012],
        [-0.0004,  0.0007,  0.0010]])

 tensor([[-0.0027,  0.0052,  0.0073],
        [ 0.0018, -0.0035, -0.0048],
        [-0.0015,  0.0030,  0.0041]])

 tensor([[-0.0142, -0.0287, -0.0338],
        [ 0.0270,  0.0546,  0.0643],
        [ 0.0093,  0.0189,  0.0222]])
```

Also, the final loss (first '*L*' in the above picture) calculated in the above picture, and the *'Autograd L'* are approximately same. We have used approximation here as pytorch is taking only four points after decimal where as my program is calculating with 14 values after decimal point.
Same is the case for gradients calculated using both methods.

2. **Time taken**

   I have observed that time taken for calculating the gradients using back propagation is comparative lesser than the naive method used during first assignment thus proving the statement that back propagation is an efficient method and avoid redundant calculation. Unfortunately, I was not able to calculate the time taken for both the methods to quantify a performance value for back propagation.

3. **Method used for calculating the minimum loss function**

   Step taken to get minimum loss function:

   - Take the values of Matrix A_i, B_i, and C_i where the loss function is minimized for a particular value of x_i.
   - Find the Average of A, B, and C for N such randomly generated input vectors.
   - Use the newly calculated A_{avg}, B_{avg} and C_{avg} to get a minimum loss function for new value of input vector x_{new} .

   However these steps have not given me the expected outcome, i.e. a generalized minimum loss function. This is either caused by not using an efficient method to calculate A, B and C for generalizing a minimum Loss function for any value to input vector x. Or we can also say that back propagation is not used for or cannot be used for generalizing a function as it is dependent on the input vector x_i to calculate the gradients.

Conclusion

By doing this assignment, I got to learn an efficient way of calculating gradients for gradient descent algorithm. The naive way which was done in the previous assignment was very time consuming, and although it taught me the basics of gradient descent, but when using in a complex model, it will be quite inefficient. In comparison back propagation is excellent for the GD algorithm. However, back propagation is very dependent on input data, thus limiting the scope of its use.

But learning about it was interesting!

Reference

- https://machinelearningmastery.com/a-gentle-introduction-to-sigmoid-function/#:~:text=The%20sigmoid%20function%20is%20also,between%20%2D%E2%88%9E%20and%20%2B%E2%88%9E
- https://towardsdatascience.com/derivative-of-the-sigmoid-function-536880cf918e
- https://stackoverflow.com/questions/36268077/overflow-math-range-error-for-log-or-exp
- https://towardsdatascience.com/a-step-by-step-implementation-of-gradient-descent-and-backpropagation-d58bda486110
- https://www.pycodemates.com/2023/02/backpropagation-and-gradient-descent-simplified.html
- https://towardsdatascience.com/a-deep-intuition-to-deep-learning-8f7220cd6579
- https://stats.stackexchange.com/questions/554764/is-backpropagation-a-fancy-way-of-saying-calculate-gradient-by-taking-partial-d
- https://stats.stackexchange.com/questions/23128/solving-for-regression-parameters-in-closed-form-vs-gradient-descent
- https://towardsdatascience.com/polynomial-regression-gradient-descent-from-scratch-279db2936fe9
- https://machinelearningmastery.com/a-gentle-introduction-to-sigmoid-function/#:~:text=The%20sigmoid%20function%20is%20also,between%20%2D%E2%88%9E%20and%20%2B%E2%88%9E
- https://mathinsight.org/scalar_triple_product#:~:text=Suggested%20background&text=The%20scalar%20triple%20product%20of,%2C%20which%20is%20a%20vector.
- https://www.analyticsvidhya.com/blog/2023/01/gradient-descent-vs-backpropagation-whats-the-difference/#:~:text=Backpropagation%20uses%20the%20chain%20rule,(s)%20as%20a%20constant.
- https://tutorial.math.lamar.edu/Solutions/CalcIII/PartialDerivatives/Prob5.aspx
- https://app.slack.com/client/T05QT4S6FV5/D05RA8MSA4S
- https://mathinsight.org/scalar_triple_product#:~:text=Suggested%20background&text=The%20scalar%20triple%20product%20of,%2C%20which%20is%20a%20vector.
- https://www.cuemath.com/algebra/multiplication-of-vectors/
- https://www.khanacademy.org/computing/computer-programming/programming-natural-simulations/programming-vectors/a/vector-magnitude-normalization