EEP 596 Computer Vision

HW #5

The purpose of this homework is to begin to explore computer vision concepts using PyTorch. You are free to work with fellow students but should turn in your own work. Specifically, you should be able to explain anything in your assignment if asked.

You should turn in your assignment as **"Assignment5.py"** and **"Report.pdf"** in Gradescope.

Be sure to consult the lecture notes (lec05b).

The questions are as follows.

1. **Chain rule.** For function f(x,y,z) = xy+z. Compute df/dz, df/dq, df/dx and df/dy. Where q =xy. Input values are x= -2 , y= 5 and z =-4. Return output in floating point.
2. **ReLU.** For input data = [-1,-2] and weights = [2,-3,-3], compute dx and dw after backpropagation if activation function is ReLU. Return output in floating point.
3. **Chain rule.** The lecture notes show the computation graph for f(w,x) = 1/(1 + exp(-($w_0x_0+w_1x_1+w_2$))). Use PyTorch's Autograd package to compute this.
    a. **(10 points)** In the lecture notes, the last three forward pass values are a=0.37, b=1.37, and c=0.73. Calculate these numbers to exact 4 decimal digits with round and return **in order of a, b, c**.
    b. **(10 points)** In the lecture notes, the backward pass values are ±0.20, ±0.39, -0.59, and -0.53. Calculate these numbers to exact 4 decimal digits with round and **return gradients in order of w_0, x_0, w_1 x_1,w_2.**
4. **Backprop.** Let f(w,x) = torch.tanh($w_0x_0+w_1x_1+w_2$). Assume the weight vector is w = [w_0=5, w_1=2], the input vector is x = [x_0=-1,x_1= 4], and the bias is w_2 =-2. Assume an MSE loss function, and ground truth of 1.
    a. **(10 points)** Use PyTorch to calculate the forward pass of the network **return f(w,x).**
    b. **(10 points)** Use PyTorch Autograd to calculate the gradients for each of the weights, and **return the gradient of them in order of w_0, w_1, and w_2.**
    c. **(10 points)** Assuming a learning rate of 0.1, update each of the weights accordingly. For simplicity, just do one iteration, and **return the updated weights in the order of w_0, w_1, and w_2.**
5. **Optimization.**
    a. **(15 points)** Implement Newton's method to find the minimum of a 2D floating-point image representing a paraboloid function returned from constructParaboloid(). Use image convolutions to compute the 1st and 2nd order derivatives with respect to x and y. Start from any arbitrary point and apply Newton's update rule to find the global minimum of the paraboloid. Return the final pixel coordinates where Newton's method converges.
    b. **(15 points)** Now implement SGD (that is, mini-batch gradient descent) to find the minimum of the same image. Show that SGD also converges to the global minimum from any arbitrary starting point. Return the final pixel coordinates where SGD converges. Put your answers to the following questions in your Report.pdf: 1) What

effect does the learning rate have on the result? 2) Can you find a learning rate parameter that causes divergence?