Report

February 23, 2023

1 Assignment 1 Report

1.0.1 1. Approach and design choices

In section 2 we need to keep track of an upstream gradient (delta) that gets funneled through the network backwards. The starting point delta = -2 * (y - y_hat) given in formula (5) can be calculated seperately then the rest can be calculated in a loop if you rearrage the order slightly to undo activation -> calculate weight & bias gradient -> calculate previous activated layer -> repeat.

In section 3 we needed to create a ML pipeline where we manually updated the weights. Most of the pipeline was structured after the weekly exercises but implementing weight decay and momentum was new. When I looked at the cifar10 dataset I used many different sanity checks. First looking at a sample to understand that it contained an image and an integer, looking at the distibution of samples across the integers and seeing that only the integers 0-9 were used and that the dataset was evenly split. Looking at the images corresponding to each integer to understand the categories. An important desicion was that I seperated out the training set with a fixed seed and calculated the mean and std of the images that were to be in that set. When preprocessing you should only use the training data to prevent information from the validation/testing data to bleed into your training set.

When implementing weight decay and momentum the documentation in torch.optim.SGD was helpful in realizing the order of operations and a hint in the discord group helped me understand a good way to keep track of the momentum using a dictionary using the name of the parameter as a key.

1.0.2 Questions

a) In section 2 we are asked to manually propagate the loss. This can be done in PyTorch with the .backwards() function.

b) The autograd.gradcheck in pytorch is a method used to check if the computed gradient of a function seems correct. I could use this method to double check my computed gradients for the activation functions.

c) The .step() function of the optimizer in PyTorch normally handles updating the weights.

d) The SGD algorithm without momentum can get stuck in local minima and take a long taime to get there. Momentum is used to make use of the history of the decent to help make better choices of how far to make the next step. The more we move in one direction the higher the momentum. Momentum is a term taken from physics and its use there is analogous to its use in ML.

e) The purpose of regularization in SGD to mitigate the influence of outliers. L2 regularization punishes extreme outliers harshly so that the algorithm will prefer a "smoother curve".

f) The parameters I chose to test were every combination of lrs = [0.01, 0.005], decays = [0, 0.001], momentums = [0, 0.99]. I wanted to have the option of momentum = 0 and decay = 0 in my sample and I wanted to try and vary each parameter with atleast two options. This meant that I tested 2^3 = 8 different sets of parameters. The best model was the model with lr=0.005, decay=0.001, and momentum=0.99. I used accuracy as a evaluation metric to select the best performing model. These were the scores on the different datasets: Train accuracy: 0.9126 Val accuracy: 0.8433 Test accuracy: 0.8405

g) With the best model selected the accuracy on the test set is about the same as the accuracy on the validation set. This gives me confidence that the performace on unseen data should be similar to the score 0.84. The performance is better on the training set (0.91). This indicates that the model hasn't generalized sufficiently. I believe training for more epochs would help with the generalization. 10 epochs are not suffucuent but I capped the training there because of time constraints.