Practical Tips on Assignment #2

Fuxin Li

Please note the following tips if you have trouble working on Assignment #2.

1) Subgradient of ReLU, the max is on each element separately. When you take derivative, the entries that are negative have a derivative of 0, the entries that are positive have a derivative of 1, the entries that are 0 have a subderivative between [0,1] (you can use any value).

2) Cross-entropy loss function. The loss function to be minimized should always be larger or equal to 0. If you have negative loss, you are missing a sign somewhere in your code.

3) Normalize your data. The standard way is to subtract the mean and divide by the standard deviation on EACH DIMENSION (across all training examples).

4) Do remember to have a sigmoid layer before you run the cross-entropy. Sigmoid+cross-entropy derivative has a simpler form if both are considered together as one layer.

Now, some other tips:

5) If your code runs very slow, you need to vectorize your code. Many operations can be written as array operations. For loops on each element don't serve you well! Try never have nested loops in your code. Maybe check this for vectorization tip:

https://www.safaribooksonline.com/library/view/python-for-data/9781449323592/ch04.html (https://www.safaribooksonline.com/library/view/python-for-data/9781449323592/ch04.html)

6) You usually don't need to tune the momentum very much. But learning rate is very important. Try to tune learning rate first, fix your momentum to 0.8 and give it a small weight decay. Tune learning rate first.

7) If your code always gives 50% accuracy (doesn't learn anything). The following things might be useful for debugging:

a) Change to a much simpler problem, e.g. XOR problem to see whether your network works there.

b) Check the norm of the gradient vector, see whether it goes down gradually. If it consistently goes up, you maybe missing some negative sign in your gradient. Check the training objective (cross-entropy loss on all the training data), if it goes down, you are doing OK, if it doesn't then you are not.

c) Check the output of the 2nd layer (hidden-to-output, before sigmoid), if it's giving you some huge number, it's likely that the gradient computation is on the wrong direction.

d) Fix one layer and only work on the other. Usually, the first layer (with a matrix gradient) is easier to get wrong. You can fix the first layer to be completely random projections (this actually has theoretical guarantees to work fairly well), and only work on the second layer (hidden-to-output), sigmoid and cross-entropy layers to see whether that works. In the code, this amounts to just refusing to update the first layer's weight after a random initialization (comment out the line that updates first layer W). If the network starts giving you something that are not random anymore, check whether you computed the gradients of the first layer wrongly.

e) Gradient is a very very delicate thing. Even if you miss a factor of 2 in a part of your gradient, it will go completely AWOL. Please check your gradients carefully. It's easy to write them wrongly. Believe me I have spent numerous nights during my Ph.D. fighting with my wrong gradient that just missed a factor of 2. However, having this error early in the class is better than shortly before a paper deadline (where you may no longer have a paper).

f) In the long run, making "unit tests" are usually a good thing. Whenever you finish writing a layer, write some small testing cases to see whether the layer is giving correct outputs when you give it correct inputs. This can make it easier to debug the code than writing a big network and have no clue where the error is.

g) The gradient of cross-entropy over sigmoid (layer 2 -> final layer) actually has a very simple form, see e.g.

https://www.willamette.edu/~gorr/classes/cs449/classify.html (https://www.willamette.edu/~gorr/classes/cs449/classify.html)

h) The constant term has a gradient of 1, which is different than other weight vectors (1)! If you are writing them separately, try avoid updating the constant as well.

i) If everything is correct after checking, make the learning rate smaller and smaller until it does not blow up any more.