Assignment #2 Tips

Fuxin Li

All Sections

1. If you are observing your output from sigmoid function is 0.0/1.0(it should always larger than 0 and smaller than 1 in theory), it is due to an underflow/overflow issue. This is because the input before your activation function is too large. To avoid this, you can try to normalize your input image data to [0,1], or [-1, 1]. Also, you can multiply a factor when you initialize your weights. Say, if you are using np.randn() to initialize your weights, you can multiply your weights by 0.1, elementwise to further shrink you weights.

Also, check the log-sum-exp trick:

https://www.xarg.org/2016/06/the-log-sum-exp-trick-in-machine-learning/

2. You do not need to stick exactly to the template, but make sure you keep the separation of different layers in different classes.

1) Each layer is only supposed to do what it does, i.e. each layer is not supposed to know which other layers it is connected to.

2) MLP is supposed to call the objects representing each layer in order to complete the forward pass and the backward pass.

3) You are supposed to call MLP to train/evaluate the network, not to write something else in main to separately evaluate it.

3. 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].

4. 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.

5. Do remember to have a sigmoid layer before you run the cross-entropy.

6. 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)

7. 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.

8. If your code always gives 50% accuracy (doesn't learn anything). Note the following things to debug:

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 also spent numerous nights during my Ph.D. fighting with my wrong gradient that just missed a factor of 2.

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) has a very simple form, as alluded to in the assignment description 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 different gradient than the others (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.

This announcement is closed for comments