Back Propagation!

Eckel, TJHSST AI2, Spring 2023

Background & Explanation

We've learned enough about perceptrons and gradient descent to be ready for the main event: writing code to train perceptron networks!

After watching the lecture this week, the following pseudocode should make sense. There are several details here worth noting:

- x and y represent vectors.
- The circled x represents item-wise multiplication.
- Dot product should come before item-wise multiplication in order of operations; numpy might not do this by default, so be free with your parentheses usage.
- Capital N refers to the number of perceptron layers in the network (not including the input layer).
- Capital T refers to transpose.

```
for each epoch:
    for each x, y in training set:
        a^0 = x
        for each layer L in network:
            dot^L = w^L·a^(L-1) + b^L
            a^L = A(dot^L)
        Δ^N = A'(dot^N)⊗(y-a^N)
        for each layer L in network (counting down from N-1):
            Δ^L = A'(dot^L) ⊗ (w^(L+1))^T·Δ^(L+1)
        for each layer L in network:
            b^L = b^L + λ·Δ^L
            w^L = w^L + λ·Δ^L·(a^(L-1))^T
```

After that, each specific task will tell you what kind of output to expect from your network and how to display it.

Challenge 1: Check your understanding

Copy the example network from the notes. Once through forward propagation, you should see that the error calculated at the end (on the single input/output pair given) is 0.047608. With a learning rate of 0.1, run one step of the back propagation process, updating the weights and biases.

After that, use the new weight and bias values on the *same* input/output pair you originally tried (which is not what the real algorithm will do, but we're doing it now just to check). If you've used the learning rate of 0.1, you should now get that the new error on that specific input/output pair is 0.045736.

If your numbers aren't correct, you have an error somewhere! I will not check this as part of your submission, but if this is wrong all the rest will be too. Feel free to message me your new weight and bias matrices and I can help tell you where the error falls (though you should look for yourself as well; I might not be able to reply immediately).

Challenge 2: Train "SUM"

The "SUM" function takes two inputs, either 1s or 0s, and gives two outputs. It adds the two given inputs, representing the 2s place and 1s place digits of the binary representation of the sum. This is the set of all possible input/output pairs:

(0, 0) → (0, 0)
(0, 1) → (0, 1)
(1, 0) → (0, 1)
(1, 1) → (1, 0)

For example, in the last row, 1 + 1 = 10 in binary.

Train a [2, 2, 2] network (2 inputs, 2 hidden perceptrons, 2 output perceptrons) to learn SUM. Use the standard sigmoid function as A(x) and find an appropriate lambda by testing.

Some details:

- Please note that while training your network, ***you should not round your output***. Each perceptron should output the value produced by the sigmoid activation function, compare it to the desired output (either 1 or 0) and define an error function from there. This means that the error will never be zero; this is intentional!
- Once the network is *trained*, then when running the network you should round the output of each output perceptron (so, each perceptron in the final layer only!) so that anything greater than or equal to 0.5 is printed as "1" and anything below is printed as "0".
- Your code should begin by choosing random values for all weights and biases between -1 and 1.

At each step in the training, output the current values of the *un-rounded output layer* (that is, a^2). You should see that they get closer and closer to 0 and to 1 (as appropriate) as the network converges – it's fun to watch! The final trained results should be 100% accurate.

Challenge 3: Do the circle properly!

Train a network to accomplish the unit circle problem. Use the standard Sigmoid activation function and, once again, note that during training the output should *not* be rounded. Use a rounding threshold of 0.5 for trained final output.

Some things to consider:

- There is a file on the course website with 10,000 random points. You can use this as a training set (classifying each point correctly using the distance formula) and loop through them repeatedly or you can just randomly generate as many points as you want and keep the loop going for whatever number of specific points you wish.
- You may wish to consider an algorithm that scales the lambda value based on the total error, making smaller steps as the error gets smaller. Experiment a bit.
- You can feel free to use the [2, 4, 1] architecture we've been using so far but this algorithm can apply just as easily to larger networks. Starting from random values on the original architecture may take several tries to converge to a good solution. Much faster results are possible on larger networks; a [2, 12, 4, 1] network was popular last year, for instance.
- Alternately, copy over your values from Perceptrons 4 and start there; see if you can improve!

The goal is to produce a network that is clearly improving as a result of back propagation. If you start from random values at the beginning, a good goal for a [2, 4, 1] network is mis-classifying less than 200 points. For a [2, 12, 4, 1] network, a good goal is mis-classifying less than 40. It's normal for different random runs to have different outcomes!

Get Your Code & Document Ready to Submit

As with last time, please **follow these instructions carefully!**

Your code again needs to have multiple functionalities. You will get one command-line input. It will be either "S" or "C".

1. If you receive "S", you should train your SUM network from scratch. After **each data point**, output specifically **the value of the output vector, without rounding, from the final layer of perceptrons**. Hardcode whatever number of epochs seems appropriate; runtime should be less than one minute.
2. If you receive "C", you should train your CIRCLE network from scratch. After **each training epoch completes**, that is to say after 10,000 individual data points, in a **separate for loop**, you should run the entire training set through your current network, as if your code had stopped training, and see how many points are incorrectly classified. **Output the epoch number and number of mis-classified points after each epoch completes**. I'm not expecting this to run all the way to completion while I grade; I will observe long enough to see that the number of mis-classified points is decreasing, and that will be enough for me.

Finally, I'd like to see the most successful run of your CIRCLE training! If you're starting from random, be sure to at least hit the goals given on the previous page; I won't be too impressed with a run that mis-classifies 1000 points! Copy the console output of the run that got to the smallest total and paste into a word or txt document. Submit your document as well.

Specification

Submit **a single python script** and a **document file** to the link on the course website.

This assignment is **complete** if:

- You follow the instructions on the submission form to format your submission properly.
- Your code and document match the specifications above.