Perceptrons, Part 2 – Training

Eckel, TJHSST AI2, Spring 2021

Background & Explanation

Last week, we laid the groundwork – now it's time to train our perceptrons and see what they can, and can't, model.

A video explaining the perceptron training algorithm is linked on the course website.

The formulas mentioned in the video are:

To calculate current output: $f^* = A(\vec{w} \cdot \vec{x} + b)$

To update the weight vector: $\vec{w} = \vec{w} + (f(\vec{x}) - f^*) \cdot \lambda \cdot \vec{x}$

To update the bias: $b = b + (f(\vec{x}) - f^*) \cdot \lambda$

Remember to keep looping until either your target number of generations has been reached OR two consecutive epochs have the same identical outcome.

If that isn't clear, please re-watch the video!

A few important notes about this algorithm:

- I wasn't clear about this in the video, but it's possible for two consecutive epochs to have identical outcomes while the accuracy still isn't 100%. It just means that any changes during that epoch get reversed by later inputs. Bottom line: **AFTER THIS PROCESS COMPLETES, YOU MUST CHECK ACCURACY SEPARATELY. You can't just assume it's 100% correct.**
- In case you're curious – this will certainly converge to a 100% correct answer *if such an answer can be found*, but it *isn't* guaranteed to converge to the *best* answer if 100% isn't available. So, we can definitively answer "which Boolean functions are 100% reproducible", but not "what is the closest we can get to a Boolean function that isn't reproducible."
- Finally, don't forget that I mention 100 epochs being sufficient in the video. Don't go higher than that; it'll inflate your runtimes unnecessarily!

Required Task

Your goal is to write code that will determine how many *n*-bit Boolean functions can be perfectly modeled by a perceptron, for (in theory) any value of *n* (though $n > 4$ is impractical). So, your code should:

1. Take a number of bits as an argument.
2. Generate every truth table possible with that number of bits (ie, loop over each possible value for canonical integer representation for that number of bits, returning the truth table for each one.)
3. For each truth table:
    a. Start a perceptron model with a zero vector for weight and a zero value for bias and run the perceptron training algorithm until it completes, as shown in the video.
    b. **TEST THE ACCURACY** of the completed perceptron by looping over the truth table one more time and calculating the percentage of input vectors that are categorized correctly by your final perceptron.
4. Output the total number of possible functions, and how many of those truth tables could be 100% correctly modeled. For example, for $n = 2$, the answer should be "16 possible functions; 14 can be correctly modeled."

Find the answers for 3 bits and 4 bits. Four bits will take several minutes to run; don't be impatient! When you're done, **send your answers to me on Mattermost OR check them with another student who already knows the right answers and then tell me you've done so.** (So if you're working when I'm not available, you can still check with someone else.)

Get Your Code & Answers Ready to Submit

Make your code receive *two command-line inputs*. The first one will represent a number of bits and the second one will specify the canonical integer representation of a specific Boolean function. For example, 4 60800 would specify 4-bit Boolean function #60800. Your code should train a perceptron using the process described on the first page and, after 100 epochs or stability is reached, output:

1. The final weight vector
2. The final bias value
3. The accuracy of the perceptron as a decimal or percent

Specification

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

This assignment is **complete** if:

- The "Name" field on the Dropbox submission form contains your **class period**, then your **last name**, then your **first name**, in that order.
- Your code matches the specification above.

Specification for Outstanding Work: Graph Your Trained Perceptrons

This is one of my favorite OW specifications for the whole year and I hope lots of people do it!

Install the **matplotlib** package if you haven't already. Find a tutorial somewhere that shows you what you need to accomplish the following. Then, write code that will loop over each 2-bit Boolean function's truth table and:

- Completely train a perceptron as described on page 1.
- Output an inequality graph on a **domain and range of (-2,2)** to show the results of your training! Your graph should have:
    - A *small* dot every 0.1 units in both directions. **Color each small dot according to whether or not its coordinates return 1 or 0 when passed through your perceptron.** (Note that this means you'll pass non-integer values into your perceptron; as shown on page 2, this should be fine – it's just a linear inequality after all!)
    - A *large* dot at each of the four actual input vectors from the truth table, just like you see in the Desmos graphs on pages 2 and 3. **The large dots should not be colored according to the perceptron output; they should be colored according to the truth table.**
- In other words, what you'll see on each graph is big green dot(s) surrounded by little green dots, and big red dot(s) surrounded by little red dots, *except* for functions #6 and #9, where we should see at least one big dot surrounded by the wrong color!

In all, your code should output 16 separate graphs.

Submit your **code** to the link on the course website.

This assignment is **complete** if:

- The "Name" field on the Dropbox submission form contains your **class period**, then your **last name**, then your **first name**, in that order.
- Your code matches the specification above. (No command line arguments; it just runs.)

Specification for Outstanding Work: Animate the Perceptron Training Process

This is an experiment; I haven't assigned this before, and I'm not entirely sure how it'll turn out. I'm interested to see if it works!

Write code that accepts **two command line arguments**. The first is the canonical integer representation of a 2-bit Boolean function. The second is a training rate, lambda (see formulas on page 1).

Your code should train a perceptron to match that particular Boolean function, outputting exactly the same thing that is described in the previous OW spec (little dots, big dots.) But: this time, you should output your graph after *every single step* in the training process (4 times per epoch). Specifically, the little dots should change color and the big dots shouldn't. As a result, we should be able to see your perceptron's solution space bounce around until it lines up with the 4 big dots correctly!

Control the animation speed so that the training process is clearly viewable with a lambda of 0.1.

(A word on packages – I don't actually know if **matplotlib** will do animation well. You may have to resort to **tkinter**, which I know will work perfectly, if a little less naturally for making graphs!)

Submit your **code** to the link on the course website.

This assignment is **complete** if:

- The "Name" field on the Dropbox submission form contains your **class period**, then your **last name**, then your **first name**, in that order.
- Your code matches the specification above. (Two command line arguments as specified above.)