Lab NN2 – Backprop 1

AI – Gabor – 9 May 2021

In this lab you are creating a neural net that implements a logic gate. The single argument to your script will be a file which specifies inputs and outputs. Each line in this file specifies an input => expected-output training pair. The inputs (always 0s and 1s) will be to the left of an "=>" while the expected outputs (always a 0 or 1) will be to the right. Within a given file, the number of inputs is the same on each line, and the number of expected outputs is the same on each line. In this lab the number of inputs will be 1, 2, 3, or 4 and only a single output is expected in each case.

The contents of an XOR specification could look like:

```
0 1 => 1
1 1 => 0
0 0 => 0
1 0 => 1
```

Your script should output a neural network specification. That means that it should specify the architecture of the network along with the weight values. The architecture is specified by the script writing

```
Layer counts: …
```

where the ellipsis is a sequence of space separated numbers indicating the number of nodes in each layer (including the input and output).

Every problem in the lab can be solved by using an architecture for the NN of:

(# of inputs + 1) 2 1 1.

In the case above that would mean

```
Layer counts: 3 2 1 1
```

The first number indicates the number of inputs and the reason it is one larger than the number of inputs shown in the spec file is called a bias, explained shortly.

Following this, the weights corresponding to your neural net structure should be output. The grader will pick up on these weights and construct the network, and test it on the same n inputs specified in the file. There will be a certain total error, which must be below ½ to get any credit (clearly, if your output was always ½ the total error would be 2^{n-3}, but that's not a very impressive network). Your entire set of weights should be put into a string and printed at the same time. I think the grader looks for the last network it can find, and if you get timed out in the middle of printing a network line by line, the grader will be unhappy. You can preclude this by printing the network as a single string (complete with newlines).

In electrical components and neural networks there is often an element called a bias. A bias just means a constant input that doesn't change (eg. a battery). In the case of our networks, if you opt for a bias (and you should), it will be 1 and is construed as an extra input wire (the final input wire). Many neural nets assume that each cell gets a bias. **Our non-standard network has at most a single bias** (of value 1) to accompany the input. If your input size declaration is one greater than the number of actual inputs, then a bias is assumed.

In your feed forward network, you have implemented a dot product. In the back propagation algorithm to compute the negative of the gradient, there is also ample opportunity to compute dot products, but you have to be more careful. The slices you use to implement the backwards dot products require more care than going forward, but slices may still help you out.

Computing the transfer function can result in some math errors (overflow or underflow), especially if you are computing the gradient incorrectly. For example, e^(very big number) might be too big for even python, and if python decides you are teasing it, it may throw up its hands and throw an error. It's your responsibility to trap for this error and do the right thing.

In the back propagation algorithm you compute the negative of the gradient, so you know the direction to head in. But how far should you head? This negative gradient is typically multiplied by a number (often designated α (alpha)). It is often in the region .01 to .1 and is a parameter you can play with in training your networks. Some people adapt α over time.

The typical training scheme is that one iterates through all given input/output pairs multiple times. Each complete iteration is known as an epoch. You could also pick input/output pairs randomly or just keep cycling through them without concerning yourself about epochs.