Problem: 1

Implement the routines listed in `neural.h`, as was discussed in class. These, together with the lattice functions provided in the `Ising` package, should enable you to create a functional Hopfield neural network. After checking that your network functions in a manner similar to what was demonstrated in class, use your network to decode the mystery characters in the `final` directory on the git repo. There are seven distorted patterns (001.dat - 007.dat) in the `mystery` subdirectory. For each of these patterns:

- Load your network with the four patterns in `character_groups` that correspond to the numbered mystery pattern.
- Evolve the network.
- This is hopefully result in convergence to the correct, undistorted pattern. Your evolved pattern should match one of the four inputs exactly, giving a Hamming distance of zero. Store or record the character you converge to.

String the seven decoded patterns together, in order. What's the mystery word?

The mystery word that I obtained was: PHYS352.

Although I was able to converge in on the correct pattern, the Hamming distance for "H" was not zero; rather, the Hamming distance for and "H" was 0.080 as shown below:

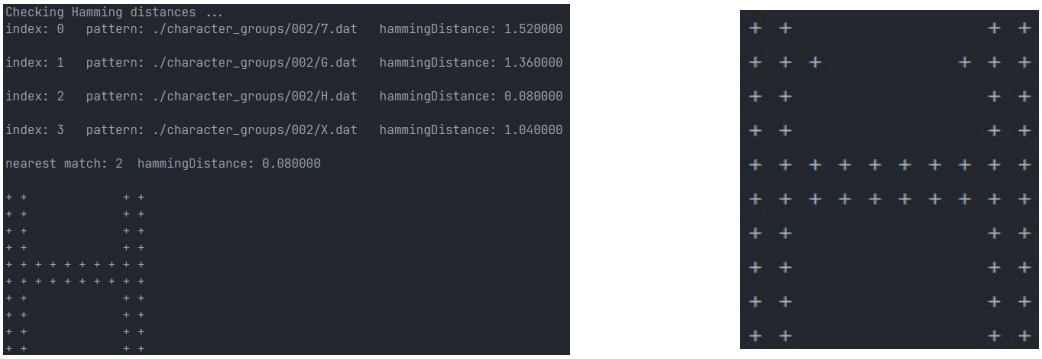

Figure 1: The Hamming Distance for "H" and the measured converged character.

After running the program several, several times, I consistently kept getting this exact Hamming distance value. One possibility that I could think of for why I was seeing such values was that the converged character that I kept getting (as shown above) was another minimum. One possible fix to this issue (I think) would be to increase the temperature; doing so would increase the variability/scope so that we can land in the minimum that we want.

I have tried to implement this change, but saw varying results. For example, when I increased the temperature slightly, I was indeed able to obtain Hamming distance values of 0 for "H" – but at the expense of other characters to have a nonzero Hamming distance value. The only other possibility is in how I implemented the code, but I currently do not see any issues with the implementation with respect to what the text and lectures suggest; however, I could be (most likely) wrong here. At the very least, we do see that the Hamming distance value for "H" is much smaller than the other Hamming distance values within that group.

The entire output of the program can be seen in `output`.

Remark: On How I Implemented `neural.h`

I have implemented the energyFunction function to return the corresponding energy value for a certain spin; this means that I kept the spin as-is, and I did not return twice the energy value. As for how the energy of a particular spin is calculated, it is using the following:

$$E_i = -\sum_j J_{i,j} s_i s_j$$

Notice that the sum is over all possible pairs of spins with respect to spin i. It is in the neuralFlip function that I introduced the factor of -2.0 attached to the energyFunction; this factor of -2.0 is what will give ΔE_{flip}. With this, the implementation of the neuralFlip is almost exactly the same as boltzmannFlip from previous assignments and lectures; there are two major differences:

1. Flipping the spin now occurs when $\Delta E_{\text{flip}} < 0$ – compared to $\Delta E_{\text{flip}} \leq 0$ in the boltzmannFlip method.
2. I had to include the situation where $T = 0$ – in which, if $\Delta E_{\text{flip}} \geq 0$, then the spin remains unchanged.

As for the getTotalEnergy method, I implemented this method to be exactly the same as the getEnergy method from assignment 6. In particular, the equation used to calculate the total energy is as follows:

$$E = -\frac{1}{2}\sum_{i,j} J_{i,j} s_i s_j = \frac{1}{2}\sum_i E_i$$

Notice that the factor of $\frac{1}{2}$ is needed in this case as the energies from each spin would be double-counted otherwise. Since we have already implemented the method to calculate E_i, all that was left to do was to calculate and sum up E_i for each spin in the lattice (while accounting for double-counting).