Parisa Suchdev
10/4/22
Hopfield Write up

I got the satisfaction of have an ANN working because I think it's working (I hope it actually is). Started working on this network with Anna Jane and then with Shaurya Swami. Anna Jane and I sat down and did it on paper first and got it working, then started with hardcoding and finally I was able to turn it to dynamic programming. Since I wrapped my head around it early, I was able to understand why Hopfield decided to update performance rules. The inputs you gave to solve were clever because I got to know why rule 2 was important (rule 1 couldn't deal with Sj values of zero.
However, the testing part was not 100% accurate and since the dataset is very small, each change in value constitutes to at least 25% of the error. Here I tested another input both on paper and code:

Input	Rule 1	Rule 2
[1, 0, 1, 1]	[1, 1, 1, 1]	[1, 1, 0, 0]

100% accurate predicted output would have been [1,0,1,1] but if not, then it should at least match to input that constitutes least error. In this case, it should have matched [1,0,1,0] to get 25% error. The output we got matches more V2 = [1,1,01], that's 50% error rather than 25%

I added the [1,0,1,1] into training set and the testing set and result was 100% accurate that means weights are playing huge part into predicting accurate values by **storing the memory** into the network! It's amazing! If it's actually how human neurons work then the reason why there is error in testing set is because of 'memory lapses' or to put into other words, if the weights are bigger/stronger then they tend to remember more and predict more accurately? Because then links are strong and even as humans, we remember things that we have strong link to.

There are the inputs for testing:

Original input: V1 = [0 0 0 0], V2 = [1 1 0 1], V3 = [1 0 1 0]

S no	Input	Rule 1	Rule2	Rule 1 Error	Rule 2 Error	Expected similarity to get 25% error
1	1 0 1 1	1 1 1 1	1 1 0 0	25%	75%	V3 = 1 0 1 0
2	1 1 1 1	1 1 0 1	1 1 0 1	25%	25%	V2 = 1 1 0 1
3	1 1 1 0	1 1 1 1	1 0 0 1	25%	75 %	V3 = 1 0 1 0
4	1 0 0 0	1 1 1 1	0 1 1 1	75%	100%	V1 = 0 0 0 0 or V3 = 1 0 1 0
5	0 0 0 1	1 1 0 1	1 1 0 0	50%	75%	V1 = 0 0 0 0
6	0 0 1 0	1 0 1 0	1 0 0 0	25%	50%	V1 = 0 0 0 0 or V3 = 1 0 1 0

Rule 1 has less error ratio than rule 2 and I do not exactly know, why? Because training set is too small? Model underfitting? Might be because I added more data and neurons into it and results look better:

```
Weight Matrix:
 [[ 0.  0.  4.  0.  0.]
 [ 0.  0. -2.  2.  2.]
 [ 4. -2.  0. -2. -2.]
 [ 0.  2. -2.  0.  2.]
 [ 0.  2. -2.  2.  0.]]
```

```
print("Original Input:\n", X)
print("Output Matrix:\n",HN.perform(X))
```

```
Original Input:
 [[0 0 0 0 0]
 [1 1 0 1 1]
 [1 0 1 0 1]
 [1 0 1 1 0]
 [1 1 1 0 0]
 [0 1 0 1 1]]
Output Matrix:
 [{'Rule1': array([1., 1., 1., 1., 1.]), 'Rule2': array([0., 0., 0., 0., 0.])}
 {'Rule1': array([1., 1., 0., 1., 1.]), 'Rule2': array([0., 1., 0., 1., 1.])}
 {'Rule1': array([1., 1., 1., 1., 0.]), 'Rule2': array([1., 0., 1., 0., 0.])}
 {'Rule1': array([1., 1., 1., 0., 1.]), 'Rule2': array([1., 0., 1., 0., 0.])}
 {'Rule1': array([1., 0., 1., 1., 1.]), 'Rule2': array([1., 0., 1., 0., 0.])}
 {'Rule1': array([1., 1., 0., 1., 1.]), 'Rule2': array([0., 1., 0., 1., 1.])}]
```

```
print("Original Input:\n", X)
print("Test Input:\n", Test)
print("Output Matrix:\n",HN.perform(Test))
```

```
Original Input:
 [[0 0 0 0 0]
 [1 1 0 1 1]
 [1 0 1 0 1]
 [1 0 1 1 0]
 [1 1 1 0 0]
 [0 1 0 1 1]]
Test Input:
 [[1 0 1 1 0]
 [1 1 1 1 1]
 [1 1 1 0 0]
 [1 0 0 0 1]
 [0 0 0 1 1]
 [0 0 1 0 0]]
Output Matrix:
 [{'Rule1': array([1., 1., 1., 0., 1.]), 'Rule2': array([1., 0., 1., 0., 0.])}
 {'Rule1': array([1., 1., 0., 1., 1.]), 'Rule2': array([1., 1., 0., 1., 1.])}
 {'Rule1': array([1., 0., 1., 1., 1.]), 'Rule2': array([1., 0., 1., 0., 0.])}
 {'Rule1': array([1., 1., 1., 1., 1.]), 'Rule2': array([0., 1., 1., 1., 0.])}
 {'Rule1': array([1., 1., 0., 1., 1.]), 'Rule2': array([0., 1., 0., 1., 1.])}
 {'Rule1': array([1., 0., 1., 0., 0.]), 'Rule2': array([1., 0., 0., 0., 0.])}]
```