Simulating Language: Lab 5 Worksheet

The simulation in `learning1.py` implements the ***learning*** of a signalling system. In the previous simulations, an individual agent's signalling system was provided innately, and didn't change in its lifetime. Populations of agents evolved through natural selection according to the fitness function we specified, to be 'optimal' in some way for communication.

In this simulation, we're ignoring evolution, and instead allowing the weights in an agent's signalling system to change through learning, as a result of their experiences.

The first section of the `learning1.py` code is similar to the code we used in our first simulation (`signalling1.py`), when we introduced the following:

- a signalling system is represented as a list of lists - you can think of this as a matrix or as a neural network.
- how to produce a signal to express a meaning, using winner-take-all;
- how to decide which meaning a received signal is expressing, using winner-take-all;
- communication as a measure of how well the speaker's meaning matches the hearer's meaning after being transmitted via a signal.

These should all be very familiar by now. We have made one major change to the code though. In `signalling1.py` we had separate matrices for production and reception. From now on we are going to use a model where we just have a single matrix which handles both processes. There are some small changes to the code to accomplish this.

> *Identify the changes required to go from a two-matrix model to a one-matrix model, and figure out why they have been made.*

Learning

```
# ----- new code below -----

def learn(system, meaning, signal):
    system[meaning][signal] += 1
```

In learning, agents store the association between the meaning and signal. We need one simple function to implement learning.The function **learn** takes three arguments and is just two lines of code. The arguments are:
1. a signalling system
2. a meaning
3. a signal

The function finds the appropriate cell in the signalling system matrix indexed by the meaning and signal, and adds one to the value of the weight in this cell.

Make sure you understand how this learning function works, what the parameters mean, and how the function updates the correct cell in the matrix.

```
In [1]: s = [[0,0,0],[0,0,0],[0,0,0],[0,0,0]]

In [2]: learn(s,0,2)

In [3]: learn(s,1,1)

In [4]: learn(s,0,2)

In [5]: learn(s,3,0)

In [6]: s
Out[6]: [[0,0,2],[0,1,0],[0,0,0],[1,0,0]]
```

Enter the code in the box and try it out.

Create a signalling system, then modify it by learning some random meaning-signal pairs.

Make sure you understand how and why the weights in the matrix have changed.

Training

```
def train(system, ms_pair_list):
    for pair in ms_pair_list:
        learn(system, pair[0], pair[1])
```

Rather than input each learning episode individually (which is a bit laborious), we can give an agent a list of meaning-signal pairs, and learn them all through the single function **train**. This function goes through each item in the list, and learns each meaning-signal pair individually.

Create a signalling system, then provide it with a list of learning exposures and check that the system has learnt from the data you gave it.

Questions

1. How good is this model of learning? How can you test it?
2. Learning is implemented as a frequency count of associations. Are there other reasonable ways of updating the matrix? How else might you change weights in response to observations?
3. Can you write some code to test how well an agent has learnt a language?