Simulating Language: Lab 8 Worksheet

Download `learning4.py` from the usual place. The simulation explores how to implement cross-situational learning, with as few changes to the code we already know as possible.

The new code starts with a set of parameter declarations:

- The `context_size` parameter specifies how many candidate meanings the learner receives in a learning episode (including the target).

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

meanings = 5            # number of meanings
signals = 5             # number of signals
context_size = 3        # context size
rule = [1, 0, 0, 0]     # learning rule
```

What kind of learning does `context_size = 1` simulate?

New Agent

The function `new_agent` creates a new agent. This is slightly modified from the equivalent function in the last simulation, in that it now takes a parameter `type` which defines the type of agent created:

- **random**: every cell in the signalling matrix is set initially to zero.
- **optimal**: every cell is set to zero except those on the leading diagonal.

```
# creates a new agent, with initial weights either all 0 (if type is given as
# "random"), or specifying an optimal system (if type is given as "optimal")
def new_agent(type):
    system = []
    for i in range(meanings):
        row = []
        for j in range(signals):
            if type == 'optimal':
                if i == j:
                    row.append(1)
                else:
                    row.append(0)
            if type == 'random':
                row.append(0)
        system.append(row)
    return system
```

Adding Context

The function `add_context` takes the speaker's intended meaning as a parameter and returns a context made up of this meaning and additional candidate meanings chosen at random from the other possible meanings to make the context the appropriate size defined by the parameter `context_size`. It works as follows:

1. Create a list of all possible meanings (`m_list`)
2. Remove the speaker's intended meaning from this list.
3. Choose `context_size-1` meanings at random from the remaining list to be the context.
4. Add the speaker's intended meaning back in to the context.
5. Return the context.

```
# add random context to target meaning m
def add_context(m):
    m_list = range(meanings)
    if m in m_list:
        m_list.remove(m)
    random.shuffle(m_list) # randomizes list of non-target meanings
    context = m_list[ : (context_size-1)] # take first however many
    context.append(m) # add back in target meaning
    return context
```

Producing Data

The function `produce_data` produces a single piece of data from the given signalling system. Each data item is no longer a meaning-signal pair, but instead a signal paired with a list of meanings (the context).

```
# produce a single data items from speaker_system
def produce_data(speaker_system):
    meaning = random.randrange(len(speaker_system))
    signal = wta(production_weights(speaker_system,meaning))
    context = add_context(meaning)
    return [context,signal]
```

Learning from Uncertain Data

The function `multiple_meaning_learn` is based on the `learn` function we've used before.

This function makes use of a couple of new Python list operators, namely **`in`** and **`not in`**. These provide an easy way to check whether or not an object is a member of a list, as shown in the code box. Make sure you understand how they work.

```
>>> x = [1, 2, 3, 4]
>>> 3 in x
True
>>> 5 in x
False
>>> 0 not in x
True
>>> 2 in x[2:]
False
```

Can you see how multiple_meaning_learn differs from learn (in learning3.py)?
Do you understand how the matrix is updated when the agent receives a signal-meaning list pair? If not, create your own blank agent (use the `new_agent` *function) and test it with some random meanings and signals.*

```
# learn a signal paired with multiple meanings
def multiple_meaning_learn(system,meaning_list,signal,rule):
    for m in range(len(system)):
        for s in range(len(system[m])):
            if m in meaning_list and s == signal: system[m][s] += rule[0]
            if m in meaning_list and s != signal: system[m][s] += rule[1]
            if m not in meaning_list and s == signal: system[m][s] += rule[2]
            if m not in meaning_list and s != signal: system[m][s] += rule[3]
```

The Simulation

The `xsl_simulation` function runs the simulation. It uses the global parameters defined at the beginning, and takes three additional parameters as follows:

`learning_episodes`:	the number of learning episodes to simulate
`trials`:	number of trials over which communicative accuracy is calculated
`report_every`:	the frequency with which communicative accuracy should be measured

```
def xsl_simulation(learning_episodes, trials, report_every):
    adult = new_agent('optimal')
    learner = new_agent('random')
    data_accumulator = []
    for i in range(learning_episodes):
        utterance = produce_data(adult)
        context = utterance[0]
        signal = utterance[1]
        multiple_meaning_learn(learner, context, signal, rule)
        if (i % report_every == 0):
            data_accumulator.append(ca_monte(adult, learner, trials))
    return [learner,data_accumulator]
```

It runs through the following steps:

1. Initialise the population, by creating an optimal adult/speaker and a blank learner/hearer.
2. For a number of times specified in `learning_episodes`:
    a. Produce some data.
    b. Learn this data.
    c. Every `report_every` trials, calculate the communicative accuracy between speaker and hearer and append to `data_accumulator`.
3. Finally, return details of the hearer's signalling system, plus the list of accumulated communicative accuracy scores.

Questions

1. Exploring the parameters.
    a. Run the simulation with the default parameters, using 1000 trials to calculate communicative accuracy and outputting data every 10 learning episodes. Plot these values on a graph.
    b. Change the size of the context, and plot the data on the graph. How does the size of the context affect learning? Are there any circumstances in which learning doesn't happen?
    c. Now increase the number of meanings and signals in an agent's signalling system, and run the simulation again. What is the relationship between the context size, the number of meanings, and the time taken to learn the signalling system?
2. The `xsl_simulation` function calls `ca_monte` to calculate the level of success in communication from speaker to hearer. Change this function call so that it calculates communicative success from hearer to speaker, and re-run the simulation.
    a. How does this change the results?
    b. Why does communicative success measured in this way sometimes go down?
3. Experiment with learning a random language and with different learning rules.
    a. Can you use cross-situational learning to construct a perfect language from random data?
    b. Do all the update rules classified by Smith (2002) as 'learners' still learn cross-situationally?
    c. Do you see any differences in learning time between different rules?
4. Adjust the code so that a learner is exposed to multiple signals at each exposure, with a context which includes the meanings of all of those signals. Is learning still possible? What might this model correspond to in real language learning?
5. What kinds of word learning event are likely to be particularly challenging for a cross-situational learner?