03 - Intro to subsymbolic - the paired module

August 20, 2016

1 Paired model

```
In [1]: ## Introduction
```

This is the module from unit 4 of LispACT-R tutorials, showing the workings of the subsymbolic system on it.

We start by importing relevant packages and ignoring warnings.

```
In [2]: import string
        import random
        import warnings

        import pyactr as actr

        warnings.simplefilter("ignore")
```

The module and environment are shown below. This simulation simulates an experiment in which twenty words are assigned to numbers 0-9 (two words per one number). Subjects had to learn number-word pairs. After the learning phase, they were shown a word (or words) and had to recall the corresponding number (or numbers). The word-number pairs are in the dictionary "self.text" in the environment. The environment process proceeds by (i) printing a word first, (ii) waiting for five seconds and printing the corresponding number. If the subject (or the model) presses the right number before the 5s limit, the environment proceeds to the next word (if there is any). The environment process can specify how many pairs should be used in how many trials.

The model will be explained in more detail below.

```
In [3]: class Environment(actr.Environment): #subclass Environment
            """
            Environment, putting a random letter on screen.
            """

            def __init__(self):
                self.text = {"bank": "0", "card": "1", "dart": "2", "face": "3", "game": "4",
                        "hand": "5", "jack": "6", "king": "7", "lamb": "8", "mask": "9",
                        "neck": "0", "pipe": "1", "quip": "2", "rope": "3", "sock": "4",
                        "tent": "5", "vent": "6", "wall": "7", "xray": "8", "zinc": "9"}
                self.run_time = 5

            def environment_process(self, number_pairs, number_trials, start_time=0):
                """
                Environment process. Random letter appears, model has to press the key corresponding to
                """
                used_text = {key: self.text[key]\
                             for key in random.sample(list(self.text), number_pairs)}
```

```
        time = start_time
        yield self.Event(time, self._ENV, "STARTING ENVIRONMENT")
        for _ in range(number_trials):
           for word in used_text:
                self.output(word, trigger=used_text[word]) #output on environment
                time += self.run_time
                yield self.Event(time, self._ENV, "PRINTED WORD %s" % word)
                self.output(used_text[word]) #output on environment
                time += self.run_time
                yield self.Event(time, self._ENV, "PRINTED NUMBER %s" % used_text[word])

class Model(object):
    """
    Model pressing the right key.
    """

    def __init__(self, env, **kwargs):
        self.m = actr.ACTRModel(environment=env, **kwargs)

        actr.chunktype("pair", "probe answer")

        actr.chunktype("goal", "state")

        self.dm = self.m.DecMem()

        retrieval = self.m.dmBuffer("retrieval", self.dm)

        g = self.m.goal("g")
        self.m.goal("g2", set_delay=0.2)
        start = actr.makechunk(nameofchunk="start", typename="chunk", value="start")
        actr.makechunk(nameofchunk="attending", typename="chunk", value="attending")
        actr.makechunk(nameofchunk="testing", typename="chunk", value="testing")
        actr.makechunk(nameofchunk="response", typename="chunk", value="response")
        actr.makechunk(nameofchunk="study", typename="chunk", value="study")
        actr.makechunk(nameofchunk="attending_target", typename="chunk", value="attending_targe
        actr.makechunk(nameofchunk="done", typename="chunk", value="done")
        g.add(actr.makechunk(typename="read", state=start))

        self.m.productionstring(name="attend_probe", string="""
        =g>
        isa     goal
        state   start
        ?visual>
        state   auto_buffering
        ==>
        =g>
        isa     goal
        state   attending
        +visual>""")

        self.m.productionstring(name="read_probe", string="""
        =g>
        isa     goal
```

```
state   attending
=visual>
isa     _visual
object  =word
==>
=g>
isa     goal
state   testing
+g2>
isa     pair
probe   =word
+retrieval>
isa     pair
probe   =word""")

self.m.productionstring(name="recall", string="""
=g>
isa     goal
state   testing
=retrieval>
isa     pair
answer  =ans
?manual>
state   free
?visual>
state   free
==>
+manual>
isa     _manual
cmd     'presskey'
key     =ans
=g>
isa     goal
state   study
~visual>""")

self.m.productionstring(name="cannot_recall", string="""
=g>
isa     goal
state   testing
?retrieval>
state   error
?visual>
state   free
==>
=g>
isa     goal
state   study
~visual>""")

self.m.productionstring(name="study_answer", string="""
=g>
isa     goal
state   study
```

```
?visual>
state   auto_buffering
==>
=g>
isa     goal
state   attending_target
+visual>""")

self.m.productionstring(name="associate", string="""
=g>
isa     goal
state   attending_target
=visual>
isa     _visual
object  =val
=g2>
isa     pair
probe   =word
?visual>
state   free
==>
=g>
isa     goal
state   start
~visual>
=g2>
isa     pair
answer  =val
~g2>""")
```

Before going through details of the model, we will show the trace. First, we have to initialize the environment and the model.

```
In [4]: environ = Environment()
        m = Model(environ, subsymbolic=True, latency_factor=0.4, decay=0.5,\
                  retrieval_threshold=-2, instantaneous_noise=0)
```

Then, we can run a simulation. We will run the simulation for only one pairand two trials. That means that in the first trial, the model will learn word-number association (and will not do anything). In the second trial, it will see the word and will try to recall the right number. We will run the model for 12s (10s are needed to run the first trial).

```
In [5]: sim = m.m.simulation(realtime=True, environment_process=environ.environment_process,\
                            number_pairs=1, number_trials=2, start_time=0)
        sim.run(12)
```

```
(0, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(0, 'PROCEDURAL', 'RULE SELECTED: attend_probe')
OUTPUT ON SCREEN
bank
END OF OUTPUT
(0.05, 'PROCEDURAL', 'RULE FIRED: attend_probe')
(0.05, 'g', 'MODIFIED')
(0.05, 'visual', 'CLEARED')
(0.05, 'PROCEDURAL', 'CONFLICT RESOLUTION')
```

```
(0.05, 'PROCEDURAL', 'NO RULE FOUND')
(0.1, 'visual', 'ATTENDED TO OBJECT')
(0.1, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(0.1, 'PROCEDURAL', 'RULE SELECTED: read_probe')
(0.15, 'PROCEDURAL', 'RULE FIRED: read_probe')
(0.15, 'g', 'MODIFIED')
(0.15, 'retrieval', 'START RETRIEVAL')
(0.15, 'g2', 'CLEARED')
(0.15, 'visual', 'CLEARED')
(0.15, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(0.15, 'PROCEDURAL', 'NO RULE FOUND')
(0.35, 'g2', 'CREATED A CHUNK: pair(answer=None, probe=bank)')
(0.35, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(0.35, 'PROCEDURAL', 'NO RULE FOUND')
(3.1056, 'retrieval', 'RETRIEVED: None')
(3.1056, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(3.1056, 'PROCEDURAL', 'RULE SELECTED: cannot_recall')
(3.1556, 'PROCEDURAL', 'RULE FIRED: cannot_recall')
(3.1556, 'g', 'MODIFIED')
(3.1556, 'visual', 'CLEARED')
(3.1556, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(3.1556, 'PROCEDURAL', 'NO RULE FOUND')
OUTPUT ON SCREEN
0
END OF OUTPUT
(5, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(5, 'PROCEDURAL', 'RULE SELECTED: study_answer')
(5.05, 'PROCEDURAL', 'RULE FIRED: study_answer')
(5.05, 'g', 'MODIFIED')
(5.05, 'visual', 'CLEARED')
(5.05, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(5.05, 'PROCEDURAL', 'NO RULE FOUND')
(5.1, 'visual', 'ATTENDED TO OBJECT')
(5.1, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(5.1, 'PROCEDURAL', 'RULE SELECTED: associate')
(5.15, 'PROCEDURAL', 'RULE FIRED: associate')
(5.15, 'g2', 'MODIFIED')
(5.15, 'g', 'MODIFIED')
(5.15, 'g2', 'CLEARED')
(5.15, 'visual', 'CLEARED')
(5.15, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(5.15, 'PROCEDURAL', 'NO RULE FOUND')
OUTPUT ON SCREEN
bank
END OF OUTPUT
(10, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(10, 'PROCEDURAL', 'RULE SELECTED: attend_probe')
(10.05, 'PROCEDURAL', 'RULE FIRED: attend_probe')
(10.05, 'g', 'MODIFIED')
(10.05, 'visual', 'CLEARED')
(10.05, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(10.05, 'PROCEDURAL', 'NO RULE FOUND')
(10.1, 'visual', 'ATTENDED TO OBJECT')
(10.1, 'PROCEDURAL', 'CONFLICT RESOLUTION')
```

```
(10.1, 'PROCEDURAL', 'RULE SELECTED: read_probe')
(10.15, 'PROCEDURAL', 'RULE FIRED: read_probe')
(10.15, 'g', 'MODIFIED')
(10.15, 'retrieval', 'START RETRIEVAL')
(10.15, 'g2', 'CLEARED')
(10.15, 'visual', 'CLEARED')
(10.15, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(10.15, 'PROCEDURAL', 'NO RULE FOUND')
(10.35, 'g2', 'CREATED A CHUNK: pair(answer=None, probe=bank)')
(10.35, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(10.35, 'PROCEDURAL', 'NO RULE FOUND')
(11.0444, 'retrieval', 'CLEARED')
(11.0444, 'retrieval', 'RETRIEVED: pair(answer=0, probe=bank)')
(11.0444, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(11.0444, 'PROCEDURAL', 'RULE SELECTED: recall')
(11.0944, 'PROCEDURAL', 'RULE FIRED: recall')
(11.0944, 'g', 'MODIFIED')
(11.0944, 'visual', 'CLEARED')
(11.0944, 'manual', 'COMMAND: presskey')
(11.0944, 'retrieval', 'CLEARED')
(11.0944, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(11.0944, 'PROCEDURAL', 'NO RULE FOUND')
(11.3444, 'manual', 'PREPARATION COMPLETE')
(11.3444, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(11.3444, 'PROCEDURAL', 'NO RULE FOUND')
(11.3944, 'manual', 'INITIATION COMPLETE')
(11.3944, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(11.3944, 'PROCEDURAL', 'NO RULE FOUND')
(11.4944, 'manual', 'KEY PRESSED: 0')
(11.4944, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(11.4944, 'PROCEDURAL', 'NO RULE FOUND')
OUTPUT ON SCREEN
0
END OF OUTPUT
(11.5944, 'manual', 'MOVEMENT FINISHED')
(11.5944, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(11.5944, 'PROCEDURAL', 'RULE SELECTED: study_answer')
(11.6444, 'PROCEDURAL', 'RULE FIRED: study_answer')
(11.6444, 'g', 'MODIFIED')
(11.6444, 'visual', 'CLEARED')
(11.6444, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(11.6444, 'PROCEDURAL', 'NO RULE FOUND')
(11.6944, 'visual', 'ATTENDED TO OBJECT')
(11.6944, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(11.6944, 'PROCEDURAL', 'RULE SELECTED: associate')
(11.7444, 'PROCEDURAL', 'RULE FIRED: associate')
(11.7444, 'g2', 'MODIFIED')
(11.7444, 'g', 'MODIFIED')
(11.7444, 'g2', 'CLEARED')
(11.7444, 'visual', 'CLEARED')
(11.7444, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(11.7444, 'PROCEDURAL', 'NO RULE FOUND')
```

1.1 Breaking the model into parts

We initialize the ACT-R model by stating "actr.ACTRModel(environment=env, **kwargs)". The keyword arguments are used for the subsymbolic system. We will come back to it in a minute.

Apart from the special way of initializing, there is nothing new in this model. We specify a dm and its buffer, a goal buffer and an imaginal buffer (called "g2"). We also create several chunks used in rules, and we specify production rules.

The crucial bit happens when initializing our model:

```
In [6]: m = Model(environ, subsymbolic=True, latency_factor=0.4, decay=0.5,\
                  retrieval_threshold=-2, instantaneous_noise=0)
```

The model is initialized with the environment "environ" and several other arguments. The other arguments are the keyword arguments used by the ACT-R model. These arguments specify parameters in the subsymbolic system. First, the subsymbolic part has to be switched on by setting "subsymbolic" to True. Then, we specify latency_factor (lf parameter in LispACT-R), decay (bll parameter in LispACT-R), retrieval threshold (rt parameter in LispACT-R), and noise (ans parameter in LispACT-R). The full list of parameters is present in a separate file, actrparameters.