# 03 - Intro to subsymbolic - the paired module

May 20, 2016

## 1 Paired model

In [1]: *## Introduction*

This is the module from unit 4 of LispACT-R tutorials. We will show 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.environment as env
        import pyactr.model as model

        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(env.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]\
```

```python
                        for key in random.sample(list(self.text), number_pairs)}

        time = start_time
        yield env.Event(env.roundtime(time), env._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 env.Event(env.roundtime(time), env._ENV,\
                            "PRINTED WORD %s" % word)
                self.output(used_text[word]) #output on environment
                time += self.run_time
                yield env.Event(env.roundtime(time), env._ENV,\
                            "PRINTED NUMBER %s" % used_text[word])


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

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

        self.m.chunktype("pair", "probe answer")

        self.m.chunktype("goal", "state")

        self.dm = self.m.DecMem()

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

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

    def attend_probe(self):
        yield {"=g": self.m.Chunk("goal", state=self.start),\
                "?visual": {"state": "auto_buffering"}}
        yield {"=g": self.m.Chunk("goal", state=self.attending), "+visual": None}

    def read_probe(self):
        yield {"=g": self.m.Chunk("goal", state=self.attending),\
                "=visual": self.m.Chunk("_visual", object="=word")}
        yield {"=g": self.m.Chunk("goal", state=self.testing),\
                "+g2": self.m.Chunk("pair", probe="=word"),\
                "+retrieval": self.m.Chunk("pair", probe="=word")}
```

```python
        def recall(self):
            yield {"=g": self.m.Chunk("goal", state=self.testing),\
                    "=retrieval": self.m.Chunk("pair", answer="=ans"),\
                    "?manual": {"state": "free"},  "?visual": {"state": "free"}}
            yield {"+manual": self.m.Chunk("_manual", cmd="presskey", key="=ans"),\
                    "=g": self.m.Chunk("goal", state=self.study), "~visual": None}

        def cannot_recall(self):
            yield {"=g": self.m.Chunk("goal", state=self.testing),\
                    "?retrieval": {"state": "error"}, "?visual": {"state": "free"}}
            yield {"=g": self.m.Chunk("goal", state=self.study), "~visual": None}

        def study_answer(self):
            yield {"=g": self.m.Chunk("goal", state=self.study),\
                    "?visual": {"state": "auto_buffering"}}
            yield {"=g": self.m.Chunk("goal", state=self.attending_target),\
                    "+visual": None}

        def associate(self):
            yield {"=g": self.m.Chunk("goal", state=self.attending_target),\
                    "=visual": self.m.Chunk("_visual", object="=val"),\
                    "=g2": self.m.Chunk("pair", probe="=word"),\
                    "?visual": {"state": "free"}}
            yield {"=g": self.m.Chunk("goal", state=self.start),\
                    "~visual": None, "=g2": self.m.Chunk("pair", answer="=val"), "~g2": None}
```

Before going through details of the model, we will show the trace. First, we have to initialize the environment and the model. We also have to specify what production rules are used in the model.

```python
In [4]: environ = Environment()
        m = Model(environ, subsymbolic=True, latency_factor=0.4, decay=0.5,\
                    retrieval_threshold=-2, instantaneous_noise=0)
        m.m.productions(m.attend_probe, m.read_probe, m.recall, m.cannot_recall,\
                        m.study_answer, m.associate)
```

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).

```python
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
pipe
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, 'g2', 'CLEARED')
(0.15, 'retrieval', 'START RETRIEVAL')
(0.15, 'visual', 'CLEARED')
(0.15, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(0.15, 'PROCEDURAL', 'NO RULE FOUND')
(0.35, 'g2', 'CREATED A CHUNK: pair(probe=pipe, answer=None)')
(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
1
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, 'g', 'MODIFIED')
(5.15, 'g2', 'MODIFIED')
(5.15, 'visual', 'CLEARED')
(5.15, 'g2', 'CLEARED')
(5.15, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(5.15, 'PROCEDURAL', 'NO RULE FOUND')
OUTPUT ON SCREEN
pipe
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, 'g2', 'CLEARED')
(10.15, 'retrieval', 'START RETRIEVAL')
(10.15, 'visual', 'CLEARED')
(10.15, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(10.15, 'PROCEDURAL', 'NO RULE FOUND')
(10.35, 'g2', 'CREATED A CHUNK: pair(probe=pipe, answer=None)')
(10.35, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(10.35, 'PROCEDURAL', 'NO RULE FOUND')
(11.0444, 'retrieval', 'CLEARED')
(11.0444, 'retrieval', 'RETRIEVED: pair(probe=pipe, answer=1)')
(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: 1')
(11.4944, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(11.4944, 'PROCEDURAL', 'NO RULE FOUND')
OUTPUT ON SCREEN
1
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, 'g', 'MODIFIED')
(11.7444, 'g2', 'MODIFIED')
(11.7444, 'visual', 'CLEARED')
(11.7444, 'g2', '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 "model.ACTRModel(environment=env, **kwargs)". Obviously, there is a new part in kwargs, we specify that the model could have keyword arguments. 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 much special 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.