Python ACT-R Tutorials
- 1. Getting Started
  - 1. Install Python
  - 2. Install CCMSuite
  - 3. Run a model
- 2. Symbolic System
  - 1 – Simple production
  - 2 – Simple DM model
  - 3 – Simple instructions
- 3. Sub Symbolic System
  - 1 – Simple forgetting
  - 2 – Simple activation
  - 3 – Simple spreading
  - 4 – Simple partial
  - 5 – Simple inhibition
  - 6 – Simple utility learning
- 4. Environments
  - 1 – Simple environment
  - 2 – Simple agents
- 5. Vision
  - 1 – Simple vision
  - 2 – Top down system
- 6. Create Modules
  - 1 – Simple modules
- Reference Material
  - Associative Memory
  - Declarative Memory
  - Deconstructing ACT-R
  - Manual/Tutorial
  - Perceptual Motor
  - Python Productions
  - Utility algorithms

Sitemap

Reference Material >

Associative Memory

A New Associative Memory System

Sun, 13/07/2008 - 22:22 — terry

As part of ongoing research, we wanted to examine the effects of context on memory retrieval. The idea is that being in certain context may make it easier (or harder) to recall certain chunks. The main mechanism in standard ACT-R for doing this is Spreading Activation: if a chunk in declarative memory shares a slot value with a chunk in a buffer, this can increase the activation of the chunk in memory, making it easier to recall. However, this approach requires chunks to be carefully organized so that the chunks with similar context contain a specific matching slot value.

As an alternate approach, we would like to have a system that learns to increase the activation of chunks in declarative memory based purely on association. That is, if we currently have `A B C` in a buffer and we do a recall that retrieves `X Y Z`, then in the future if we have `A B C` in that buffer, then `X Y Z` should have its activation increased so that it will be easier to recall. This is an experience-based priming mechanism, as opposed to the semantic priming of spreading activation.

To implement this, we need some formula to determine how to increase the activation based on these shared retrievals. For now, we are using the standard base level learning formula.

To demonstrate it, the following model is shown a series of words and is asked to look at each word in turn and recall the name of its offspring, if any. If we assume that the previous word is kept in some sort of working memory (usually implemented in ACT-R as the `imaginal` buffer), then the previous word will act as a context to adjust the activation of working memory chunks that are often used in that situation. It should be noted that this is probably a horrible model of this situation: this model is only meant for demonstration purposes.

First, we implement the Environment. The sequence of words to be shown is given by `['blue','cat','red','dog','blue','cat','red','dog']`. Note that this should mean that recalling that the cat's offspring is a kitten will get faster whenever it is preceded by `blue`, and a similar process should happen for `dog`,`puppy`, and `red`. We measure the reaction time to evaluate this increase in speed.

```
import ccm
log=ccm.log(html=True)

class Environment(ccm.Model):
    def start(self):
        self.message=None
        yield 1
        for item in ['blue','cat','red','dog','blue','cat','red','dog']:
            self.message=item
            x=self.now()
            yield self.say     # wait until something is said
            log.rt=self.now()-x
            yield 0.1
            self.message=None
            yield 0.25

    def say(self,word):
        self.said_word=word
```

Next, the agent. It should be noted that we are fixing the baselevel activation of the two memory chunks to be 1. This reflects the fact that these are supposed to be highly accessible frequently used chunks with many years of experience (as opposed to a fact that we're memorizing for the purposes of the experiment). This means that the normal base level learning equation is not being used, so the activation of this chunk will not change via the baselevel equation: any changes in reaction time will be due to the associative memory system.

```
from ccm.lib.actr import *
class AssociativeModel(ACTR):
    goal=Buffer()
    imaginal=Buffer()

    retrieval=Buffer()
    memory=Memory(retrieval,threshold=0)
    DMBaseLevel(memory)
    DMAssociate(memory,imaginal,weight=0.1)

    def init():
        memory.add('child cat kitten',baselevel=1)
        memory.add('child dog puppy',baselevel=1)
        goal.set('read')


    def read_word(goal='read',top='message:?word!None'):
        goal.set('respond ?word')

    def get_response(goal='respond ?word',memory='busy:False'):
        memory.request('child ?word ?')
        goal.set('say ?word')

    def say_response(goal='say ?word',retrieval='child ?word ?word2'):
        self.parent.say(word2)
        goal.set('imagine')

    def say_response_fail(goal='say',memory='error:True'):
        self.parent.say('unknown')
        goal.set('imagine')

    def imagine(goal='imagine',top='message:?word!None'):
        imaginal.set(word)
        goal.set('wait')

    def wait(goal='wait',top='message:None'):
        goal.set('read')
```

The associative memory system is activated by the line `DMAssociate(memory,imaginal,weight=0.1)`. If this line is removed, no association will occur. The first parameter indicates the `Memory` module to associate with, the second parameter indicates the `Buffer` to use as the context, and the `weight` sets how strong the association is (i.e. how much to adjust the activation, with 1.0 meaning to give this the same strength as the normal baselevel activation formula).

If this model is run, we see that the second recall of each chunk is faster than the first. We can check the context-specificity by adjusting the context. That is, if we give the sequence `['blue','cat','red','dog','blue','cat','red','cat']` instead, then there will be no boost to the final recall.

Comments

You do not have permission to add comments.