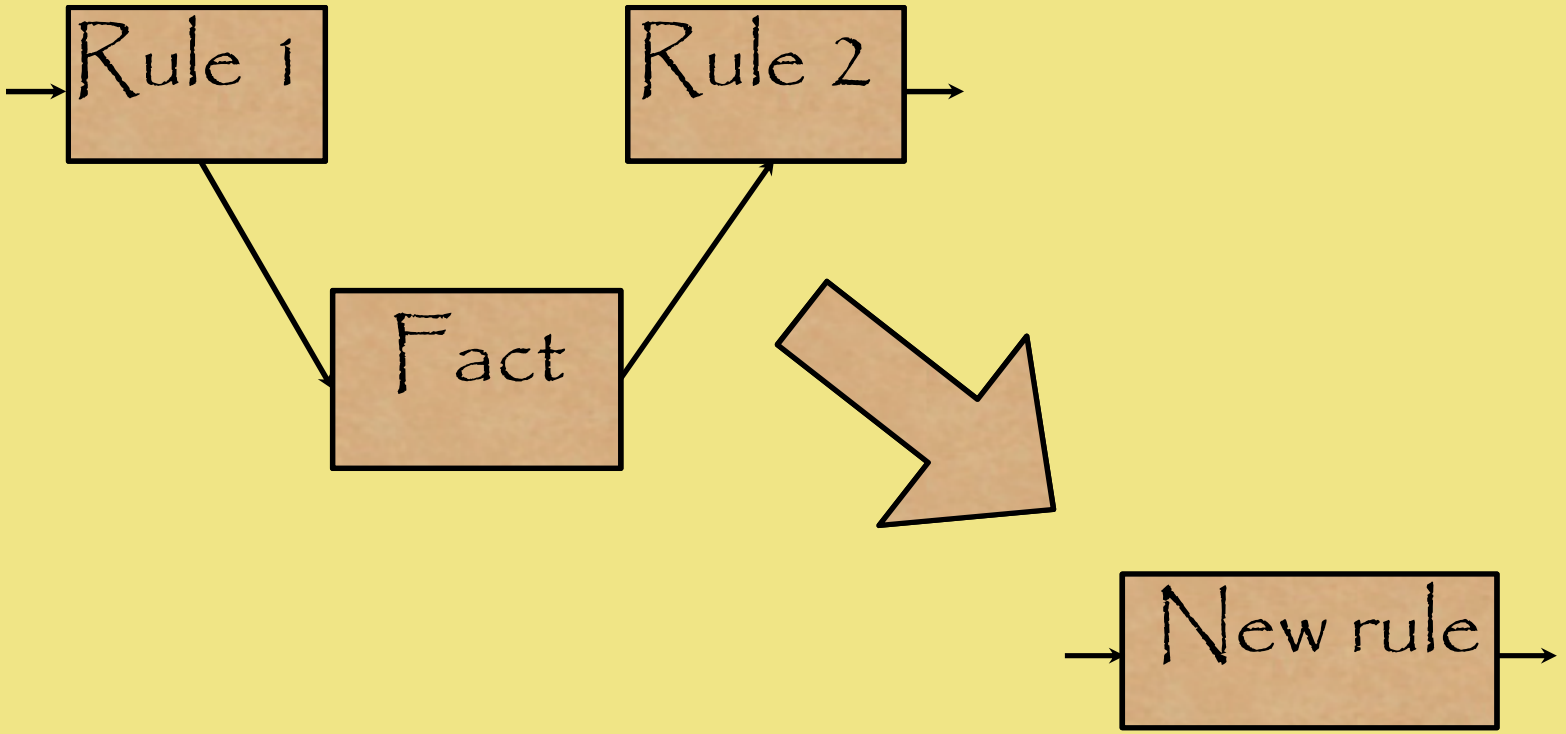
Learning new production rules

Niels Taatgen

Carnegie Mellon

rijksuniversiteit groningen

Combination and Specialization



Production compilation: specialization and combination

- Production rules are specialized by factoring out the retrieval request and retrieval match (if any, otherwise there is no specialization)
- Production rules are combined by merging two rules into one

Support for skill acquisition

- By factoring out retrievals
  - Speedup because retrievals are no longer needed
  - It reduces errors (retrieval errors)
  - Frees up “working memory” by reducing interference

Discontinuous learning

- In declarative memory, activation (how often is a chunk used) determines its success
- In procedural memory, utility (how efficient is the use of knowledge) determines its success
- A shift from declarative to procedural may produce transition effects: for example, U-shaped learning

Example: Addition

```
(P initialize-
  addition
=goal>
  ISA     add
  arg1    =num1
  arg2    =num2
  sum     nil
==>
=goal>
  sum     =num1
  count   zero
+retrieval>
  ISA  order
  first   =num1)
```

```
orderx
isa order
first six
second seven
```

```
(P increment-sum
=goal>
   ISA  add
   sum   =sum
   count =count
=retrieval>
   ISA order
   first =sum
   second =newsum
==>
=goal>
  sum =newsum
+retrieval>
   ISA order
   first =count)
```

Example: Addition

```
(P initialize-
  addition
=goal>
  ISA      add
  arg1     six
  arg2     =num2
  sum      nil
==>
=goal>
  sum      six
  count    zero
+retrieval>
  ISA  order
  first    six)
```

```
orderx
isa order
first six
second seven
```

```
(P increment-sum
=goal>
   ISA  add
   sum   six
   count =count
orderx>
   ISA order
   first six
   second seven
==>
=goal>
  sum seven
+retrieval>
   ISA order
   first =count)
```

Example: Addition

```
(P initialize-
  addition
=goal>
  ISA     add
  arg1    six
  arg2    =num2
  sum     nil
==>
=goal>
  sum     six
  count   zero
+retrieval>
  ISA  order
  first   six)
```

```
(P increment-sum
=goal>
   ISA  add
   sum   six
   count =count
orderx>
   ISA order
   first six
   second seven
==>
=goal>
  sum seven
+retrieval>
   ISA order
   first =count)
```

Example: Addition

```
(P initialize-
  addition
=goal>
  ISA     add
  arg1    six
  arg2    =num2
  sum     nil
==>
=goal>
  sum     six
  count   zero
+retrieval>
  ISA  order
  first   six)
```

```
(P increment-sum
=goal>
   ISA  add
   sum   six
   count zero
orderx>
   ISA order
   first six
   second seven
==>
=goal>
  sum seven
+retrieval>
   ISA order
   first zero)
```

Example: Addition

```
(P initialize-
  addition
=goal>
  ISA     add
  arg1    six
  arg2    =num2
  sum     nil
==>
=goal>
  sum     six
  count   zero
+retrieval>
  ISA  order
  first   six)
```

```
(P new-rule
=goal>
   ISA     add
   arg1    six
   arg2    =num2
   sum     nil
==>
=goal>
   sum seven
   count zero
+retrieval>
   isa order
   first zero)
```

```
(P increment-sum
=goal>
   ISA  add
   sum   six
   count zero
orderx>
   ISA order
   first six
   second seven
==>
=goal>
   sum seven
+retrieval>
   ISA order
   first zero)
```

What happens to other buffers?

- As long as there are no conflicts, buffer conditions and actions are all copied into the new rule
- Conflicts:
  - First rule has an action that second rule uses as a condition
    - e.g. first rule +visual>, second rule =visual>
  - Both rules have an action on the same buffer
    - e.g. first rule +manual>, second rule also +manual>

Example: Paired Associate

```
(p read-probe
    =goal>
      isa       goal
      state     attending
      arg1      nil
    =visual>
      isa       text
      value     =val
==>
    +retrieval>
      isa       goal
      state     associated
      arg1      =val
    =goal>
      arg1      =val
      state     testing
)
```

```
zinc-9
   isa goal
   state associated
   arg1 zinc
   arg2 “9”
```

```
(p recall
    =goal>
      isa       goal
      arg1      =val
      state     testing
    =retrieval>
      isa       goal
      arg1      =val
      arg2      =ans
    ?manual>
      state free
==>
    +manual>
      isa       press-key
      key       =ans
    =goal>
      state     read-study-item
)
```

Example: Paired Associate

```
(p read-probe
    =goal>
      isa       goal
      state     attending
      arg1      nil
    =visual>
      isa       text
      value     zinc
==>
    +retrieval>
      isa       goal
      state     associated
      arg1      zinc
    =goal>
      arg1      zinc
      state     testing
)
```

```
zinc-9
    isa goal
    state associated
    arg1 zinc
    arg2 “9”
```

```
(p recall
    =goal>
      isa       goal
      arg1      zinc
      state     testing
    zinc-9>
      isa       goal
      arg1      zinc
      arg2      “9”
    ?manual>
      state free
==>
    +manual>
      isa       press-key
      key       “9”
    =goal>
      state     read-study-item
)
```

Carnegie Mellon

rijksuniversiteit groningen

Example: Paired Associate

```
(p read-probe
    =goal>
      isa       goal
      state     attending
      arg1      nil
    =visual>
      isa       text
      value     zinc
==>
    +retrieval>
      isa       goal
      state     associated
      arg1      zinc
    =goal>
      arg1      zinc
      state     testing
)
```

```
(p recall
    =goal>
      isa       goal
      arg1      zinc
      state     testing
    zinc-9>
      isa       goal
      arg1      zinc
      arg2      "9"
    ?manual>
      state free
====>
    +manual>
      isa       press-key
      key       "9"
    =goal>
      state     read-study-item
)
```

Carnegie Mellon

rijksuniversiteit groningen

Example: Paired Associate

```
(p read-probe
    =goal>
      isa       goal
      state     attending
      arg1      nil
    =visual>
      isa       text
      value     zinc
==>
    +retrieval>
      isa       goal
      state     associated
      arg1      zinc
    =goal>
      arg1      zinc
      state     testing
)
```

```
(p zinc-9-rule
    =goal>
      isa       goal
      state     attending
      arg1      nil
    =visual>
      isa       text
      value     zinc
    ?manual>
      state free
==>
    =goal>
      arg1      zinc
      state     read-study
    item
    +manual>
      isa       press-key
      key       "9"
)
```

```
(p recall
    =goal>
      isa       goal
      arg1      zinc
      state     testing
    zinc-9>
      isa       goal
      arg1      zinc
      arg2      "9"
    ?manual>
      state free
==>
    +manual>
      isa       press-key
      key       "9"
    =goal>
      state     read-study-item
)
```

Parameter Learning: what we want from it

- Gradual introduction of new rules: after the first opportunity for the rule to be learned, it should take some more practice or experience before it will regularly be used
- Evaluation of new rules
  - If the new rule is better than the parents, it should eventually fire whenever it matches
  - If the new rule is worse than the parents, it should eventually not fire anymore

Utility of new rules

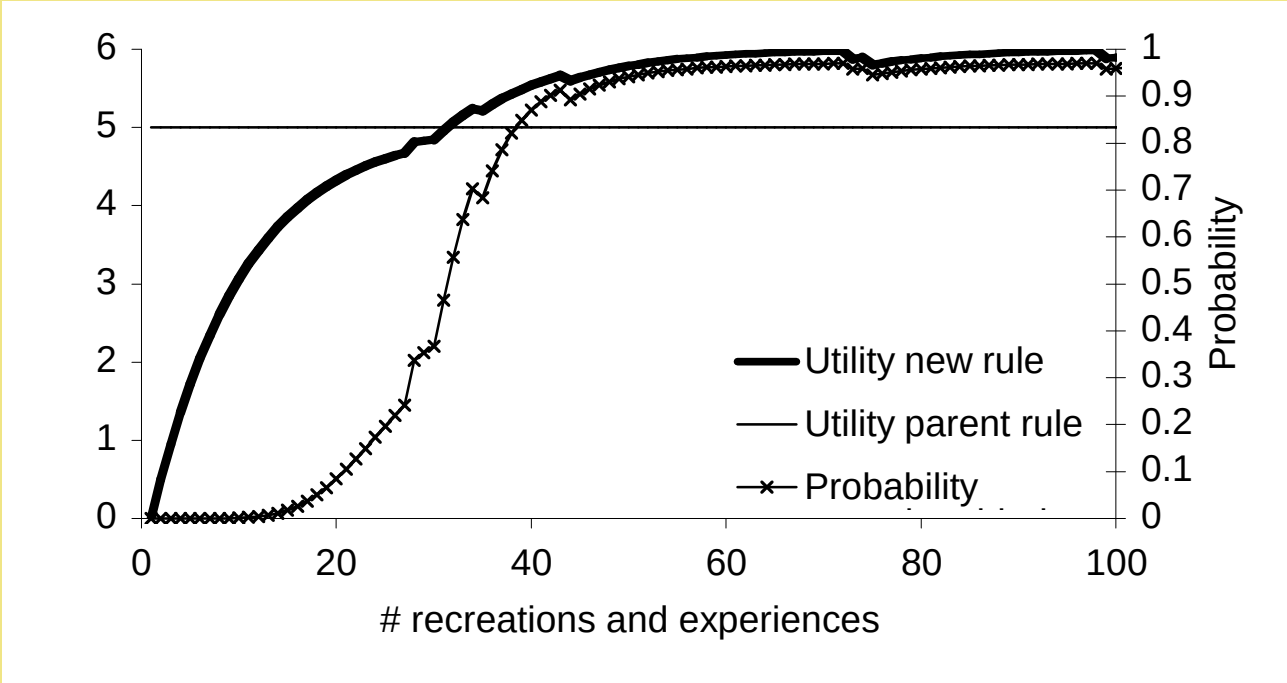
- Utility is learned according to:

$$U_i(n) = U_i(n-1) + \alpha[R_i(n) - U_i(n-1)]$$

- A new rule initially receives a Utility of zero. Each time it is recreated, its Utility is updated:

$$U_i(n) = U_i(n-1) + \alpha[U_{parent}(n-1) - U_i(n-1)]$$

Parameter learning: example

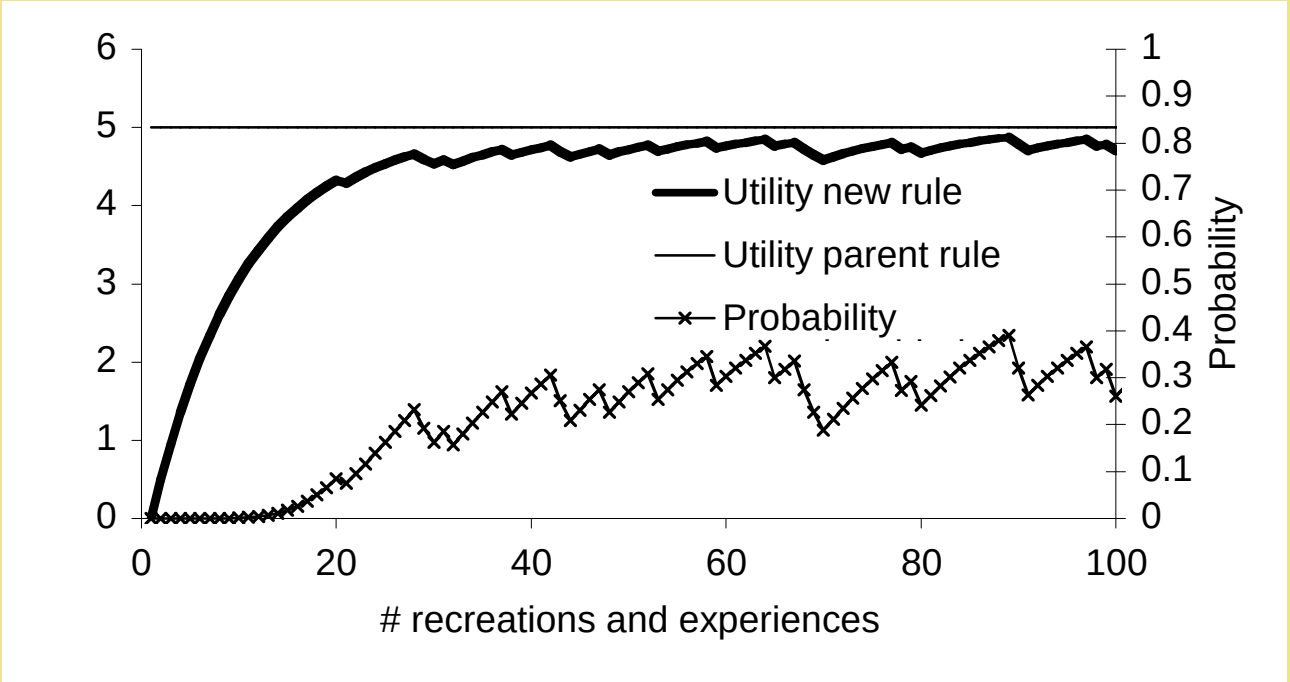


Noise = 0.2 α=0.1 Utility new rule = 6

Carnegie Mellon

rijksuniversiteit groningen

Parameter learning: example



Noise = 0.2 α=0.1 Utility new rule = 4

Carnegie Mellon

rijksuniversiteit groningen

Properties

- It takes a while for the new rule to be learned
- Rules that are recreated more often are learned faster

Instructions for paired associate

- start Read the word stimulus-read
- stimulus-read Retrieve associate recalled
- recalled Test success of recall (response found/wait)
- response found Type response wait
- wait Read feedback new trial
- new trial Complete task start

Learning from instructions

Instructions for Paired Associate Task:

(op1 isa operator pre start action read arg1 fill post stimulus-read)
(op2 isa operator pre stimulus-read action associate arg1 filled arg2 fill post recalled)
(op3 isa operator pre recalled action test-arg2 arg1 respond arg2 wait)
(op4 isa operator pre respond action type arg2 response post wait)
(op5 isa operator pre wait action read arg2 fill post new-trial)
(op6 isa operator pre new-trial action complete-task post start)

Example productions

```
(p retrieve-operator
   =goal>
       isa task
       state =state
       step ready
==>
   +retrieval>
       isa operator
        pre =state
    =goal>
       step retrieving-
   operator)
```

```
(p type-arg2
  =goal>
     isa task
     step retrieving-operator
      =imaginal>
        isa args
     arg2 =val
  =retrieval>
     isa operator
     action type
     arg2 response
     post =state
  ?manual>
    state    free
==>
   +manual>
    isa      press-key
    key      =val
  =goal>
    state    =state
    step     ready)
```

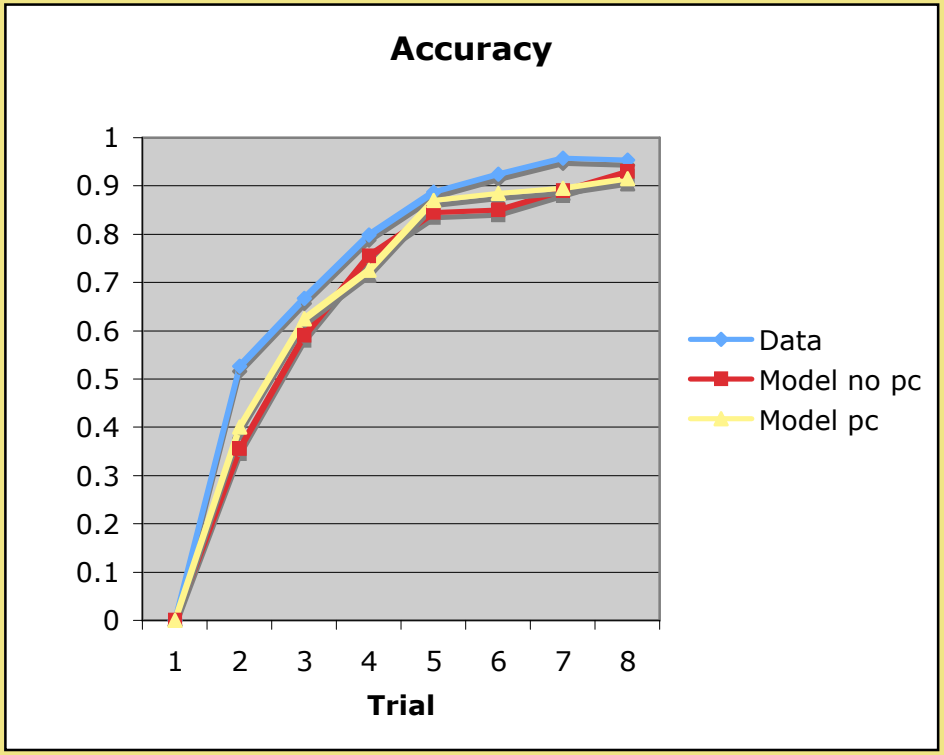
(op4 isa operator task assoc pre respond action type arg2 response post wait)

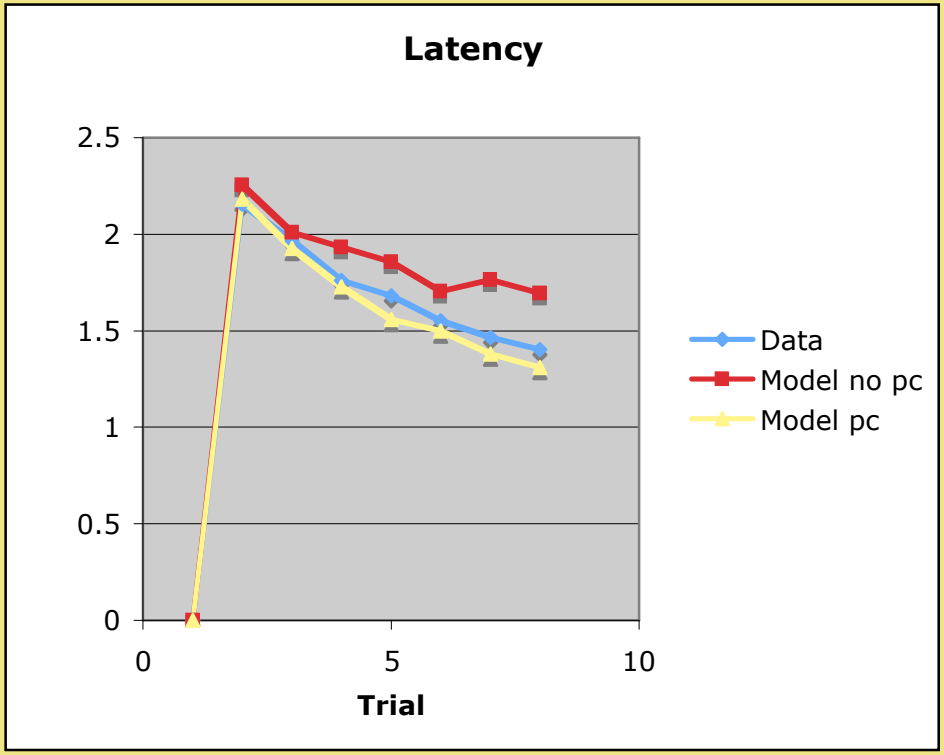
Compiled production

```
(p production323
   =goal>
     isa task
     task assoc
     state respond
     step ready
    =imaginal>
     isa args
     arg2 =val
   ?manual
     state free
==>
   +manual>
     isa press-key
     key =val
    =goal>
       state wait)
```

(op4 isa operator
task assoc
pre respond
action type
arg2 response
post wait)

Model results





Carnegie Mellon

rijksuniversiteit groningen

Unit Assignment: Learning the Past Tense

- Children go through three stages in learning the past tense
  - Stage 1: If they inflect an irregular verb, they do it correctly, e.g., break-broke
  - Stage 2: Children occasionally overgeneralize, e.g., break-breaked
  - Stage 3: Children do it correct again: break-broke
- This is called U-shaped learning
  - Important aspect: children do not get feedback on whether or not the past tense they produce is correct.

Learning the past tense

- This is a different model than you have seen up to now:
  - It is not an experiment (but real life!)
  - It's on a different time scale (months instead of hours)
  - It doesn't use the perceptual and motor components of ACT-R

What is already there

The model code

generate a goal like

```
goal1
  isa goal
  state nil
```

And put a present tense in the imaginal buffer like:

```
verb23
  isa past-tense
  verb have
  stem nil
  suffix nil
```

- Your model has to fill in the stem and suffix slots in the imaginal buffer (in this case *had* and *blank*, respectively), and set the state slot in the goal to *done*.
- Three rules that are already provided in the model will then "pronounce" the word.
- The feedback the model uses is not failure or success, as kids also do not get feedback, but the costs of producing the past tense.

Three Evaluation Productions

Reward 5

```
   =goal>
     isa goal
     state done
   =imaginal>
     isa past-tense
     verb =word
     suffix blank
==>
   =goal> state nil
```

Reward 4.2

```
 =goal>
    isa goal
    state done
  =imaginal>
     isa past-tense
     verb =stem
     stem =stem
     suffix =suffix
  - suffix blank
==>
     =goal> state nil
```

Reward 3.9

```
=goal>
    isa goal
    state done
  =imaginal>
     isa past-tense
     stem nil
     suffix nil
==>
   =goal> state nil
```

What is already there

The model also assumes you perceive things in the environment (e.g., parents). So for each past tense the model creates itself, two examples are added to declarative memory:

```
word2323
  isa past-tense
  verb have
  stem had
  suffix blank
```

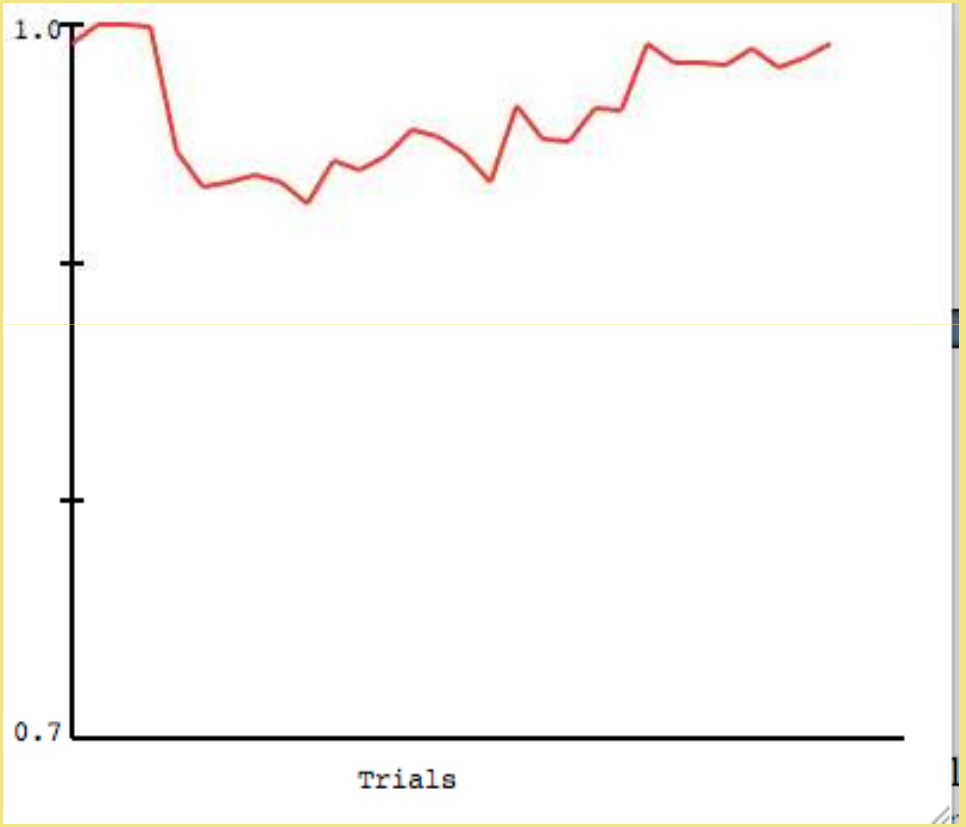
```
word4234
  isa past-tense
  verb use
  stem use
  suffix ed
```

What your model should do

- It should attempt to retrieve past tenses from memory
- Alternatively, it can try to retrieve a different past tense and do some pattern matching to apply it to the current past tense
- If something goes wrong it should just give up, and leave the stem and suffix slots empty. This represents the case where the child just used the present tense.

What your model should learn

- Your model should learn the regular rule
- Your model should exhibit U-shaped learning



Need 2 Retrieval Productions

```
(p retrieve1
   ….
==>
    +retrieval>
      isa past-tense
      - suffix nil
    ….)
```

```
(p retrieve2
    ….
==>
    +retrieval>
      isa past-tense
      verb =word
      - suffix nil
    …)
```

Challenge for the class: Why?