Unit 4: Activation of Chunks and Base-Level Learning

There are two goals of this unit. The first is to introduce the subsymbolic quantity of activation associated with chunks. The other is to show how those activation values are learned through the history of usage of the chunks.

4.1 Introduction

We have seen retrieval requests in productions like this:

```
(P example-counting
   =goal>
      isa          count
      state        counting
      number        =num1
   =retrieval>
      isa          count-order
      first       =num1
      second      =num2
==>
   =goal>
      number        =num2
   +retrieval>
      isa          count-order
      first       =num2
)
```

In this case an attempt is being made to retrieve a count-order chunk with a particular number (bound to **=num2**) in its **first** slot. Up to now we have been working with the system at the symbolic level. If there was a chunk that matched that retrieval request it would be placed into the **retrieval** buffer, and if not then the retrieval request would fail and the state of the declarative memory module would indicate an error. The system was deterministic and we did not consider any timing cost associated with that retrieval or the possibility a chunk in declarative memory might fail to be retrieved. For the simple tasks we have looked at so far that was sufficient.

Most psychological tasks however are not that simple and issues such as accuracy and latency over time or across different conditions are measured. For modeling these more involved tasks one will typically need to use the subsymbolic components of ACT-R to accurately model and predict human performance. For the remainder of the tutorial, we will be looking at the subsymbolic components that control the performance of the system. To use the subsymbolic components we need to turn them on by setting the :esc parameter (enable subsymbolic computations) to **t**:

```
(sgp :esc t)
```

That setting will be necessary for the rest of the models in the tutorial.

4.2 Activation

Every chunk in ACT-R's declarative memory has associated with it a numerical value called its activation. The activation reflects the degree to which past experiences and current context indicate that chunk will be useful at any particular moment. When a retrieval request is made the chunk with the greatest activation among those that match the specification of the request will be the one placed into the **retrieval** buffer. There is one constraint on that however. There is a parameter called the retrieval threshold which sets the minimum activation a chunk can have and still be retrieved. It is set with the :rt parameter:

```
(sgp :rt -0.5)
```

If the chunk with the highest activation among those that match the request has an activation which is less than the retrieval threshold, then no chunk will be placed into the **retrieval** buffer and an error state will indicate the failure.

The activation A_i of a chunk i is computed from three components – the base-level, a context component and a noise component. We will discuss the context component in the next unit. So, for now the activation equation is:

$$A_i = B_i + \varepsilon$$

B_i**:** The base-level activation. This reflects the recency and frequency of practice of the chunk i.

ε: The noise value. The noise is composed of two components: a permanent noise associated with each chunk and an instantaneous noise computed at the time of a retrieval request.

We will discuss these components in detail below.

4.3 Base-level Learning

The equation describing learning of base-level activation for a chunk i is:

$$B_i = \ln(\sum_{j=1}^{n} t_j^{-d})$$

n**:** The number of presentations for chunk i.
t_j: The time since the *jth* presentation.

d: The decay parameter which is set using the :bll (base-level learning) parameter. This parameter is almost always set to 0.5.

This equation describes a process in which each time an item is presented there is an increase in its base-level activation, which decays away as a power function of the time since that presentation. These decay effects are summed and then passed through a logarithmic transformation.

There are two types of events that are considered as presentations of a chunk. The first is its initial entry into declarative memory. The other is any merging of that chunk with a chunk that is already in declarative memory. The next two subsections describe those events in more detail.

4.3.1 Chunks Entering Declarative Memory

When a chunk is initially entered into declarative memory is counted as its first presentation. There are two ways for a chunk to be entered into declarative memory, both of which have been discussed in the previous units. They are:

- Explicitly by the modeler using the **add-dm** command. These chunks are entered at the time the call is executed, which is time 0 for a call in the body of the model definition.

- When the chunk is cleared from a buffer. We have seen this happen in many of the previous models as visual locations, visual objects, and goal chunks are cleared from their buffers they can then be found among the chunks in declarative memory.

4.3.2 Chunk Merging

Something we have not seen previously is what happens when the chunk cleared from a buffer is an identical match to a chunk which is already in declarative memory. If a chunk has the same chunk-type and all of its slot values are an exact match with one of the existing chunks in declarative memory then instead of being added to declarative memory it goes through a process we refer to as merging with that existing chunk. Instead of adding the new chunk to declarative memory the preexisting chunk in declarative memory is credited with a presentation. Then the name of the new chunk (the one which is being merged into declarative memory) is changed to now reference the chunk that was already in declarative memory i.e. there is one chunk which now has two (or possibly more) names. This mechanism results in repeated completions of the same operations (the clearing of a duplicate chunk) reinforcing the chunk that represents that situation instead of creating lots of identical chunks each with only one presentation. So, for example, repeatedly attending the same visual stimuli would result in strengthening a single chunk that represents that object.

4.4 Optimized Learning

Because of the need to separately calculate the effect of each presentation, the learning rule is computationally expensive and for some models the real time cost of computation is too great to be able to actually run the model in a reasonable amount time. To reduce the computational cost there is an approximation that one can use when the presentations are approximately uniformly distributed over the time since the item was created. This approximation can be enabled by turning on the optimized learning parameter - :ol. In fact, its default setting is on (the value **t**). When optimized learning is enabled, the following equation applies:

$$B_i = \ln(n/(1-d)) - d * \ln(L)$$

n: The number of presentations of chunk i.
L: The lifetime of chunk i (the time since its creation).
d: The decay parameter.

4.5 Noise

The noise component of the activation equation contains two sources of noise. There is a permanent noise which can be associated with a chunk and an instantaneous noise value which will be recomputed at each retrieval attempt. Both noise values are generated according to a logistic distribution characterized by a parameter s. The mean of the logistic distribution is 0 and the variance, σ^2, is related to the s value by this equation:

$$\sigma^2 = \frac{\pi^2}{3} s^2$$

The permanent noise s value is set with the :pas parameter and the instantaneous noise s value is set with the :ans parameter. Typically, we are only concerned with the instantaneous noise (the variance from trial to trial) and leave the permanent noise turned off (a value of **nil**).

4.6 Probability of Recall

If we make a retrieval request and there is a matching chunk, that chunk will only be retrieved if it exceeds the retrieval activation threshold, τ. The probability of this happening depends on the expected activation, A_i, and the amount of noise in the system which is controlled by the parameter s:

$$recall\ probability_i = \frac{1}{1 + e^{\frac{\tau - A_i}{s}}}$$

Inspection of that formula shows that, as A_i tends higher, the probability of recall approaches 1, whereas, as τ tends higher, the probability decreases. In fact, when $\tau = A_i$, the probability of recall is .5. The *s* parameter controls the sensitivity of recall to changes in activation. If *s* is close to 0, the transition from near 0% recall to near 100% will be abrupt, whereas when *s* is larger, the transition will be a slow sigmoidal curve.

4.7 Retrieval Latency

The activation of a chunk also determines how quickly it can be retrieved. When a retrieval request is made, the time it takes until the chunk that is retrieved is available in the **retrieval** buffer is given by this equation:

$$Time = Fe^{-A}$$

A: The activation of the chunk which is retrieved.
F: The latency factor parameter. This parameter is set using the :lf parameter.

If no chunk matches the retrieval request, or no chunk has an activation which is greater than the retrieval threshold then a retrieval failure will occur. The time it takes for the failure to be signaled is:

$$Time = Fe^{-\tau}$$

τ: The retrieval threshold.
F: The latency factor.

4.8 The Paired-Associate Example

Now that we have described how activation works, we will look at an example model which shows the effect of base-level learning. Anderson (1981) reported an experiment in which subjects studied and recalled a list of 20 paired associates for 8 trials. The paired associates consisted of 20 nouns like "house" associated with the digits 0 - 9. Each digit was used as a response twice. Below is the mean percent correct and mean latency to type the digit for each of the trials. Note subjects got 0% correct on the first

trial because they were just studying them for the first time and the mean latency is 0 only because there were no correct responses.

Trial	Accuracy	Latency
1	.000	0.000
2	.526	2.156
3	.667	1.967
4	.798	1.762
5	.887	1.680
6	.924	1.552
7	.958	1.467
8	.954	1.402

The Unit 4 folder contains the paired model for this experiment. The experiment code is written to allow one to run a general form of the experiment. Both the number of pairs to present and the number of trials to run can be specified. You can run the model through n trials of m paired associates (m no greater than 20) by calling paired-task with those parameters:

```
(paired-task m n)
```

If you would like to do the task as a person, include the symbol human as the third parameter to paired-task:

```
(paired-task m n 'human)
```

For each of the m words you will see the stimulus for 5 seconds during which you have the opportunity to make your response. Then you will see the associated number for 5 seconds. The simplest form of the experiment is one in which a single pair is presented twice. Here is the trace of the model doing such a task. The first time the model has an opportunity to learn the pair and the second time it has a chance to recall that learned pair:

```
> (paired-task 1 2)
 0.000   GOAL        SET-BUFFER-CHUNK GOAL GOAL REQUESTED NIL
 0.000   VISION      SET-BUFFER-CHUNK VISUAL-LOCATION VISUAL-LOCATION0-0 REQUESTED NIL
 0.000   PROCEDURAL  CONFLICT-RESOLUTION
 0.050   PROCEDURAL  PRODUCTION-FIRED ATTEND-PROBE
 0.050   PROCEDURAL  CLEAR-BUFFER VISUAL-LOCATION
 0.050   PROCEDURAL  CLEAR-BUFFER VISUAL
 0.050   PROCEDURAL  CONFLICT-RESOLUTION
 0.135   VISION      Encoding-complete VISUAL-LOCATION0-0-0 NIL
 0.135   VISION      SET-BUFFER-CHUNK VISUAL TEXT0
 0.135   PROCEDURAL  CONFLICT-RESOLUTION
 0.185   PROCEDURAL  PRODUCTION-FIRED READ-PROBE
 0.185   PROCEDURAL  CLEAR-BUFFER VISUAL
 0.185   PROCEDURAL  CLEAR-BUFFER IMAGINAL
 0.185   PROCEDURAL  CLEAR-BUFFER RETRIEVAL
 0.185   DECLARATIVE START-RETRIEVAL
```

```
 0.185   PROCEDURAL  CONFLICT-RESOLUTION
 0.385   IMAGINAL    SET-BUFFER-CHUNK IMAGINAL PAIR0
 0.385   PROCEDURAL  CONFLICT-RESOLUTION
 3.141   DECLARATIVE RETRIEVAL-FAILURE
 3.141   PROCEDURAL  CONFLICT-RESOLUTION
 3.191   PROCEDURAL  PRODUCTION-FIRED CANNOT-RECALL
 3.191   PROCEDURAL  CLEAR-BUFFER VISUAL
 3.191   VISION      CLEAR
 3.191   PROCEDURAL  CONFLICT-RESOLUTION
 3.241   PROCEDURAL  CONFLICT-RESOLUTION
 5.000   ------      Stopped because time limit reached
 5.000   VISION      SET-BUFFER-CHUNK VISUAL-LOCATION VISUAL-LOCATION1-0 REQUESTED NIL
 5.000   PROCEDURAL  CONFLICT-RESOLUTION
 5.050   PROCEDURAL  PRODUCTION-FIRED DETECT-STUDY-ITEM
 5.050   PROCEDURAL  CLEAR-BUFFER VISUAL-LOCATION
 5.050   PROCEDURAL  CLEAR-BUFFER VISUAL
 5.050   PROCEDURAL  CONFLICT-RESOLUTION
 5.135   VISION      Encoding-complete VISUAL-LOCATION1-0-0 NIL
 5.135   VISION      SET-BUFFER-CHUNK VISUAL TEXT1
 5.135   PROCEDURAL  CONFLICT-RESOLUTION
 5.185   PROCEDURAL  PRODUCTION-FIRED ASSOCIATE
 5.185   PROCEDURAL  CLEAR-BUFFER IMAGINAL
 5.185   PROCEDURAL  CLEAR-BUFFER VISUAL
 5.185   VISION      CLEAR
 5.185   PROCEDURAL  CONFLICT-RESOLUTION
 5.235   PROCEDURAL  CONFLICT-RESOLUTION
10.000   ------      Stopped because time limit reached
10.000   VISION      SET-BUFFER-CHUNK VISUAL-LOCATION VISUAL-LOCATION2-0 REQUESTED NIL
10.000   PROCEDURAL  CONFLICT-RESOLUTION
10.050   PROCEDURAL  PRODUCTION-FIRED ATTEND-PROBE
10.050   PROCEDURAL  CLEAR-BUFFER VISUAL-LOCATION
10.050   PROCEDURAL  CLEAR-BUFFER VISUAL
10.050   PROCEDURAL  CONFLICT-RESOLUTION
10.135   VISION      Encoding-complete VISUAL-LOCATION2-0-0 NIL
10.135   VISION      SET-BUFFER-CHUNK VISUAL TEXT2
10.135   PROCEDURAL  CONFLICT-RESOLUTION
10.185   PROCEDURAL  PRODUCTION-FIRED READ-PROBE
10.185   PROCEDURAL  CLEAR-BUFFER VISUAL
10.185   PROCEDURAL  CLEAR-BUFFER IMAGINAL
10.185   PROCEDURAL  CLEAR-BUFFER RETRIEVAL
10.185   DECLARATIVE START-RETRIEVAL
10.185   PROCEDURAL  CONFLICT-RESOLUTION
10.385   IMAGINAL    SET-BUFFER-CHUNK IMAGINAL PAIR1
10.385   PROCEDURAL  CONFLICT-RESOLUTION
11.145   DECLARATIVE RETRIEVED-CHUNK PAIR0-0
11.145   DECLARATIVE SET-BUFFER-CHUNK RETRIEVAL PAIR0-0
11.145   PROCEDURAL  CONFLICT-RESOLUTION
11.195   PROCEDURAL  PRODUCTION-FIRED RECALL
11.195   PROCEDURAL  CLEAR-BUFFER RETRIEVAL
11.195   PROCEDURAL  CLEAR-BUFFER MANUAL
11.195   PROCEDURAL  CLEAR-BUFFER VISUAL
11.195   MOTOR       PRESS-KEY 9
11.195   VISION      CLEAR
11.195   PROCEDURAL  CONFLICT-RESOLUTION
11.245   PROCEDURAL  CONFLICT-RESOLUTION
11.445   PROCEDURAL  CONFLICT-RESOLUTION
11.495   PROCEDURAL  CONFLICT-RESOLUTION
11.595   MOTOR       OUTPUT-KEY #(9 2)
11.595   PROCEDURAL  CONFLICT-RESOLUTION
11.745   PROCEDURAL  CONFLICT-RESOLUTION
15.000   ------      Stopped because time limit reached
15.000   VISION      SET-BUFFER-CHUNK VISUAL-LOCATION VISUAL-LOCATION3-0 REQUESTED NIL
15.000   PROCEDURAL  CONFLICT-RESOLUTION
```

```
15.050   PROCEDURAL   PRODUCTION-FIRED DETECT-STUDY-ITEM
15.050   PROCEDURAL   CLEAR-BUFFER VISUAL-LOCATION
15.050   PROCEDURAL   CLEAR-BUFFER VISUAL
15.050   PROCEDURAL   CONFLICT-RESOLUTION
15.135   VISION       Encoding-complete VISUAL-LOCATION3-0-0 NIL
15.135   VISION       SET-BUFFER-CHUNK VISUAL TEXT3
15.135   PROCEDURAL   CONFLICT-RESOLUTION
15.185   PROCEDURAL   PRODUCTION-FIRED ASSOCIATE
15.185   PROCEDURAL   CLEAR-BUFFER IMAGINAL
15.185   PROCEDURAL   CLEAR-BUFFER VISUAL
15.185   VISION       CLEAR
15.185   PROCEDURAL   CONFLICT-RESOLUTION
15.235   PROCEDURAL   CONFLICT-RESOLUTION
20.000   ------       Stopped because time limit reached
```

The basic structure of the screen processing productions should be familiar by now. The one thing to note is that because this model must wait for stimuli to appear on screen it takes advantage of the buffer stuffing mechanism so that it can wait for the change instead of continuously checking. The way it does so is that the first production that will fire, for either the probe or the associated number, has a **visual-location** buffer test on its LHS which will only match once buffer stuffing places a chunk into the buffer. Here are the **attend-probe** and **detect-study-item** productions for reference:

```
(p attend-probe
   =goal>
     isa      goal
     state    start
   =visual-location>
     isa      visual-location
   ?visual>
    state     free
  ==>
   +visual>
     isa      move-attention
     screen-pos =visual-location
   =goal>
     state    attending-probe
)

(p detect-study-item
   =goal>
     isa      goal
     state    read-study-item
   =visual-location>
     isa      visual-location
   ?visual>
     state    free
==>
   +visual>
     isa      move-attention
     screen-pos =visual-location
```

8

```
   =goal>
     state    attending-target
)
```

Because the buffer is cleared automatically by strict harvesting and no later productions issue a request for a visual-location these productions must wait for buffer stuffing to put a chunk into the **visual-location** buffer before they can match. Since none of the other productions match in the mean time the model will essentially just wait for the screen to change before doing anything else.

Now we will focus on the productions which are responsible for forming the association and retrieving the chunk. When the model attends to the probe with the **read-probe** production two actions are taken (in addition to the updating of the goal state):

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

It makes a request to the **imaginal** buffer to create a chunk of type pair which will hold the value read from the screen in the probe slot. It also makes a request through the **retrieval** buffer to retrieve a pair chunk from declarative memory which has that same probe value.

We will come back to the retrieval request shortly. For now we will focus on the creation of the pair chunk. This request will cause the imaginal module to create a new chunk which it will place into the **imaginal** buffer. This chunk will encode the association between the probe and the answer which is presented later. The **associate** production fires after the model reads the number which is associated with the probe:

```
(p associate
   =goal>
     isa       goal
     state     attending-target
   =visual>
```

```
      isa      text
      value    =val
   =imaginal>
      isa      pair
==>
   =imaginal>
      answer   =val
   -imaginal>
   =goal>
      state    start
   +visual>
      isa      clear
)
```

This production sets the answer slot of the pair chunk which is in the **imaginal** buffer to the answer which was read from the screen. It also then clears that chunk from the buffer so that it is entered into declarative memory. That will result in a chunk like this being added to the model's declarative memory:

```
PAIR0-0
  ISA PAIR
   PROBE  "zinc"
   ANSWER  "9"
```

This chunk serves as the memory of this trial. An important thing to note is that the chunk in the buffer is not added to the model's declarative memory until that buffer is cleared. Often that happens when the model later harvests that chunk from the buffer, but in this case the model does not harvest the chunk later so it is explicitly cleared at that point. One could imagine adding additional productions which would rehearse that chunk, but for the demonstration model that is not done.

This production also makes a clear request to the **visual** buffer to stop attending to the item. That is done so that the model does not perform the automatic re-encoding when the screen is updated.

Now, consider the retrieval request in the read-probe production again:

```
   +retrieval>
     isa      pair
     probe    =val
```

The declarative memory module will attempt to retrieve a pair chunk with the requested probe. Depending on whether a chunk can be retrieved, one of two production rules may apply corresponding the either the successful retrieval of such a chunk or the failure to retrieve a matching chunk:

```
(p recall
    =goal>
      isa       goal
      state     testing
    =retrieval>
      isa       pair
      answer    =ans
    ?manual>
      state     free
==>
    +manual>
      isa       press-key
      key       =ans
    =goal>
      state     read-study-item
    +visual>
      isa       clear
)

(p cannot-recall
    =goal>
      isa       goal
      state     testing
    ?retrieval>
      state     error
   ==>
    =goal>
      state     read-study-item
    +visual>
      isa       clear
)
```

The probability of the recall production firing and the mean latency for the recall will be determined by the activation of the chunk that is retrieved and will increase with repeated presentations and harvested retrievals.

The model gives a pretty good fit to the data as illustrated below in a run of 100 simulated subjects (because of stochasticity results are more reliable if there are more runs and to generate that many runs in a reasonable amount of time one must turn off the trace and remove the seed parameter to allow for differences from run to run):

```
> (paired-experiment 100)

Latency:
CORRELATION:  0.998
MEAN DEVIATION:  0.097
Trial   1       2       3       4       5       6       7       8
        0.000   2.139   1.834   1.665   1.546   1.440   1.377   1.306
```

```
Accuracy:
CORRELATION:  0.994
MEAN DEVIATION:  0.043
Trial   1       2       3       4       5       6       7       8
        0.000   0.554   0.769   0.855   0.904   0.932   0.955   0.960
```

4.9 Parameter estimation

To get the model to fit the data requires not only writing a plausible set of productions which can accomplish the task, but also setting the ACT-R parameters that control the behavior as described in the equations governing the operation of declarative memory. Running the model with the default values for the parameters produces the following results:

```
> (paired-experiment 5)


Latency:
CORRELATION:  0.000
MEAN DEVIATION:  1.619
Trial   1       2       3       4       5       6       7       8
        0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000


Accuracy:
CORRELATION:  0.000
MEAN DEVIATION:  0.777
Trial   1       2       3       4       5       6       7       8
        0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000
```

which shows a complete failure to retrieve any of the facts. Just lowering the retrieval threshold so that they can be retrieved results in something like this:

```
> (paired-experiment 5)


Latency:
CORRELATION:  0.972
MEAN DEVIATION:  1.187
Trial   1       2       3       4       5       6       7       8
        0.000   3.512   3.871   3.359   2.793   2.473   2.265   2.136


Accuracy:
CORRELATION:  0.914
MEAN DEVIATION:  0.176
Trial   1       2       3       4       5       6       7       8
        0.000   0.100   0.600   1.000   1.000   1.000   1.000   1.000
```

It shows some of the general trends, but does not fit the data well. The behavior of this model and the one that you have to write really depends on the settings of four parameters. Here are those parameters and their settings in this model. The retrieval threshold is set at -2. This determines how active a chunk has to be to be retrieved 50%

of the time. The instantaneous activation noise is set at 0.5. This determines how quickly probability of retrieval changes as we move past the threshold. The latency factor is set at 0.4. This determines the magnitude of the activation effects on latency. Finally, the decay rate for base-level learning is set to the value 0.5 which is where we recommend it be set for most tasks that involve the base-level learning mechanism.

How to determine those values can be a tricky process because the equations are all related and thus they cannot be independently manipulated for a best fit. Typically some sort of searching is required, and there are many ways to accomplish that. For the tutorial models there will typically be only one or two parameters that you will need to adjust and we recommend that you work through the process "by hand" adjusting the parameters individually to see the effect that they have on the model. There are other ways of determining parameters that can be used, but we will not be covering any in the tutorial.

4.10 The Activation trace

The parameter :act is also set in the sgp call of the **paired** model. This is the activation trace parameter. If it is turned on, it causes the declarative memory system to print the details of the activation computations that occur during a retrieval request in the trace. If you set it to **t** and reload the model and run for two trials of one pair (like the trace above) you will find these additional details where the retrievals occur:

```
     0.185   DECLARATIVE              START-RETRIEVAL
No matching chunk found retrieval failure
     0.185   PROCEDURAL               CONFLICT-RESOLUTION
     ...
     3.141   DECLARATIVE              RETRIEVAL-FAILURE
     ...
    10.185   DECLARATIVE              START-RETRIEVAL
Chunk PAIR0-0 matches
Computing activation for chunk PAIR0-0
Computing base-level
Starting with blc: 0.0
Computing base-level from 1 references (5.185)
  creation time: 5.185 decay: 0.5  Optimized-learning: T
base-level value: -0.111571856
Total base-level: -0.111571856
Adding transient noise -0.7636829
Adding permanent noise 0.0
Chunk PAIR0-0 has an activation of: -0.87525475
Chunk PAIR0-0 has the current best activation -0.87525475
Chunk PAIR0-0 with activation -0.87525475 is the best
    ...
    11.145   DECLARATIVE              RETRIEVED-CHUNK PAIR0-0
```

You may find this detailed accounting of the activation computation useful in debugging your models or just in understanding how the system computes activation values.

4.11 The :ncnar Parameter

There is one final parameter being set in the model which you have not seen before -
:ncnar (normalize chunk names after run). This parameter toggles whether or not the
system cleans up the references to merged chunk names after running a model or
potentially during the model running as described in the next unit. If the parameter is set
to **t**, which is the default, then the system will ensure that all of the slots of all the chunks
in the model which contain a chunk reference the "true name" of the chunk in the slot i.e.
the name of the original chunk in DM with which the copies have been merged. That
operation can make debugging easier for the modeler because all the slot values will be
consistent with the chunks shown to be in DM. However, if a model generates a lot of
chunks it can take a significant amount of time to maintain that consistency. Thus it can
be beneficial to turn this parameter off by setting it to **nil** when model debugging is
complete and one just wants to run for data generation or when the time to run a model
really matters. For most of the models in the tutorial, leaving it enabled will not result in
a significant run time increase, and it can be ignored. However, for this model the time to
run it can be significantly longer if it is left enabled. Thus it is set to **nil** in the starting
model code.

4.12 Unit Exercise: Alpha-Arithmetic

The following data were obtained by N. J. Zbrodoff on judging alphabetic arithmetic
problems. Participants were presented with an equation like A + 2 = C and had to
respond yes or no whether the equation was correct based on counting in the alphabet –
the preceding equation is correct, but B + 3 = F is not.

She manipulated whether the addend was 2, 3, or 4 and whether the problem was true or
false. She had 2 versions of each of the 6 kinds of problems (3 addends x 2 responses)
each with a different letter (a through f). She then manipulated the frequency with which
problems were studied in sets of 24 trials:

- In the Control condition, each of the 2, 3 and 4 addend problems occurred twice.
- In the Standard condition, the 2 addend problems occurred three times, the 3 addend
  problems twice, and the 4 addend problems once.
- In the Reverse condition, the 2 addend problems occurred once, the 3 addend
  problems twice, and the 4 addend problems three times.

Each participant saw problems based on one of the three conditions. There were 8
repetitions of a set of 24 problems in a block (192 problems), and there were 3 blocks for
576 problems in all. The data presented below are in seconds to judge the problems true
or false based on the block and the addend. They are aggregated over both true and false
responses:

```
Control Group (all problems equally frequently)
          Two     Three    Four
Block 1   1.840   2.460   2.820
Block 2   1.210   1.450   1.420
Block 3   1.140   1.210   1.170
```

```
Standard Group (smaller problems more frequent)
          Two     Three    Four
Block 1   1.840   2.650    3.550
Block 2   1.060   1.450    1.920
Block 3   0.910   1.080    1.480

Reverse Group (larger problems more frequent)
          Two     Three    Four
Block 1   2.250   2.530    2.440
Block 2   1.470   1.460    1.100
Block 3   1.240   1.120    0.870
```

The interesting phenomenon concerns the interaction between the effect of the addend and amount of practice. Presumably, the addend effect originally occurs because subjects have to engage in counting, but latter they come to rely mostly on retrieval of answers they have stored from previous computations.

The task for this unit is to develop a model of the control group data. Functions to run the experiment and most of a model that can perform the task are provided in the model called **zbrodoff**. The model as given does the task by counting through the alphabet and numbers "in its head" (using the subvocalize action of the speech module to produce reasonable timing data) to arrive at an answer which it compares to the initial equation to determine how to respond. Here is the performance of this model on the task:

```
> (zbrodoff 1)
CORRELATION:  0.289
MEAN DEVIATION:  1.309

              2 (64)        3 (64)        4 (64)
Block  1  2.301 (64)   2.806 (64)   3.287 (64)
Block  2  2.290 (64)   2.804 (64)   3.301 (64)
Block  3  2.286 (64)   2.797 (64)   3.290 (64)
```

It is always correct (64 out of 64 for each cell) but does not get any faster from block to block because it always uses the counting strategy. Your first task is to extend the model so that it attempts to remember previous instances of the trials. If it can remember the answer it does not have to resort to the counting strategy and can respond much faster.

The model encodes each trail in a chunk of type problem which has the results of its counting for the trial. A completed problem for a trial where the stimulus was "A+2 = C" would look like this:

```
PROBLEM0-0
  ISA PROBLEM
   ARG1  "a"
   ARG2  "2"
   RESULT  "c"
```

The result slot contains the result of counting 2 letters from A. An important thing to note is that the actual target letter for the trial is stored in the goal buffer for comparison after the model has finished counting to a result. The model only encodes the result of

the counting in the problem chunks. Thus the same problem chunk will result from a trial where the stimulus presented is "A+2 = D". The assumption is that the person is actually learning the letter counting facts and not just memorizing the stimulus-response pairings for the task. There will be one of those problem chunks for each of the additions which is encountered, which will be a total of six after it completes the first set of trials.

After your model is able to utilize a retrieval strategy along with the counting strategy given, your next step is to adjust the parameters so that the model's performance better fits the experimental data. The results should look something like this after you have the retrieval strategy working:

```
CORRELATION:  0.929
MEAN DEVIATION:  0.656


            2 (64)       3 (64)       4 (64)
Block  1  1.265 (64)   1.444 (64)   1.338 (64)
Block  2  1.094 (64)   1.077 (64)   1.093 (64)
Block  3  1.043 (64)   1.047 (64)   1.039 (64)
```

The model is still always responding correctly on all trials, the correlation is good, but the deviation is quite high because the model is too fast overall. The model's performance will depend on the same four parameters as the paired associate model: latency factor, activation noise, base-level decay rate, and retrieval threshold. In the model you are given, the first three are set to the same values as in the paired associate model and represent reasonable values for this task. You should not have to adjust any of those. However, the retrieval threshold (the :rt parameter) is set to its default value of 0. This is the parameter you should manipulate to improve the fit to the data. Here is our fit to the data adjusting only the retrieval threshold:

```
> (zbrodoff 20)
CORRELATION:  0.987
MEAN DEVIATION:  0.174


            2 (64)       3 (64)       4 (64)
Block  1  1.870 (64)   2.179 (64)   2.558 (64)
Block  2  1.376 (64)   1.520 (64)   1.632 (64)
Block  3  1.215 (64)   1.282 (64)   1.359 (64)
```

This experiment is more complicated than the ones that you have seen previously. It runs continuously for many trials and the learning that occurs across trials is important. Thus the model cannot treat each trial as an independent event and be reset before each one as has been done for the previous units. While writing your model and testing the fit to the data you will probably want to test it on smaller runs than the whole task. There are four functions you can use to test the experiment.

The first function to run the task is **zbrodff** which takes one parameter indicating the number of times to run the full experiment. That function will average the results of running the full experiment multiple times and report the correlation and deviation to the experimental data. It may take a long time to run, especially if you request a lot of runs for comparing to the data (to speed up the runs once you are sure the model is doing the

right thing you should definitely turn off the trace with the :v parameter in the sgp command).

You can use the **zbrodoff-block** function, which takes one optional parameter, to run 1 block of the experiment and display the results. The optional parameter controls whether or not to show the window while the task is running. If you do not supply the parameter a virtual window is used, and if you specify the parameter as **t** then a real window will be shown. The **zbrodoff-set** function is similar to **zbrodoff-block**, except it only runs through the set of 24 items once.

The **zbrodoff-trial** function can be used to run a single trial. It takes four parameters which are all single character strings and an optional fifth parameter like **zbrodoff-block** and **zbrodoff-set** to indicate whether or not to show the display. The first three are the elements of the equation to present i.e. "a" "2" "c" to present a + 2 = c. The fourth is the correct key which should be pressed for the trial, "K" for a true probe and "D" for a false probe. One thing to note is that the only one of those functions which calls **reset** is **zbrodoff**. So if you are using the other functions while testing the model keep in mind that unless you call the **reset** function or press the "Reset" button on the Control Panel, the model will remember everything it has done since the last time it was reset.

There is one additional command in the provided starting model that you have not seen before:

```
(set-all-base-levels 100000 -1000)
```

This sets the base-level activation of all the chunks in declarative memory that exist when it is called (which are the sequence chunks provided) to very large values by setting the parameters **n** and **L** of the optimized base-level equation for each one. The first parameter, 100000, specifies **n** and the second parameter, -1000, specifies the creation time of the chunk. This ensures that the initial chunks which encode the sequencing of numbers and letters maintain a very high base-level activation and do not fall below the retrieval threshold over the course of the task. The assumption is that counting and the order of the alphabet are very well learned tasks for the model and the human participants and the use of those skills does not lead to any significant learning for those things during the course of the experiment.

Another thing to notice is that the :ncnar parameter is set to **nil** in the starting model. Like the **paired** model, if that parameter is set to **t** then it will take significantly longer to run this model. You may want to set it to **t** to help with debugging your model as you develop it, but you will probably want to set it back to **nil** once you start running the model to compare its fit to the data.

Finally, because this model runs a lot of Lisp code to perform the experiment you will see improved performance in the real time it takes to run the model if the Lisp code is compiled. Some Lisp systems do this automatically (Macintosh Common Lisp for example does) but most do not. Therefore, if you would like to improve the running time of the model i.e. the real time it takes to run the model through the experiment not the

simulated time the model reports, then you should compile the file if your Lisp does not do so automatically. Loading the model through the Environment will not compile the file. If you have a Lisp with a GUI then one of the File menu options is likely "Compile and Load". Using that to load the model will result in the file being compiled and will improve performance. One important thing to keep in mind is that if you change the model file you will have to use the "Compile and Load" option again to load the file after you save the changes. The "Reload" button on the Environment's Control Panel and the reload command in ACT-R will not automatically recompile the file if you change it. They will continue to load the last compiled version, which will not reflect the changes you have made. Compiling the model files is something that you will probably want to do for the remainder of the models in the tutorial.

Anderson, J.R. (1981). Interference: The relationship between response latency and response accuracy. *Journal of Experimental Psychology: Human Learning and Memory, 7*, 326-343.

Zbrodoff, N. J. (1995). Why is 9 + 7 harder than 2 + 3? Strength and interference as explanations of the problem-size effect. *Memory & Cognition, 23* (6)*, 689-700.