Unit 5: Activation and Context

The goal of this unit is to introduce the components of the activation equation that reflect the context of a declarative memory retrieval.

5.1 Spreading Activation

The first context issue we will consider is called spreading activation. The chunks in the buffers provide a context in which to perform a retrieval. Those chunks spread activation to the chunks in declarative memory based on the contents of their slots. Those slot contents spread an amount of activation based on their relation to the other chunks, which we call their strength of association. This essentially results in increasing the activation of those chunks which are related to the current context.

The equation for the activation A_i of a chunk *i* including spreading activation is defined as:

$$A_i = B_i + \sum_k \sum_j W_{kj} S_{ji} + \varepsilon$$

Measures of Prior Learning, B_i: The base-level activation reflects the recency and frequency of practice of the chunk as described in the previous unit.

Across all buffers: The elements *k* being summed over are the buffers which have been set to provide spreading activation.

Sources of Activation: The elements *j* being summed over are the chunks which are in the slots of the chunk in buffer *k*.

Weighting: W_{kj} is the amount of activation from source *j* in buffer *k*.

Strengths of Association: S_{ji} is the strength of association from source *j* to chunk *i*.

ε: The noise value as described in the last unit.

The weights of the activation spread, W_{kj}, default to an even distribution from each slot within a buffer. The total amount of source activation for a buffer will be called W_k and is settable for each buffer. The W_{kj} values are then set to W_k / n_k where n_k is the number of slots which contain chunks in the chunk in buffer *k*.

The strength of association, S_{ji}, between two chunks *j* and *i* is 0 if chunk *j* is not the value of a slot of chunk *i* and *j* and *i* are not the same chunk. Otherwise, it is set using this equation:

$$S_{ji} = S - \ln(fan_j)$$

S: The maximum associative strength (set with the :mas parameter)

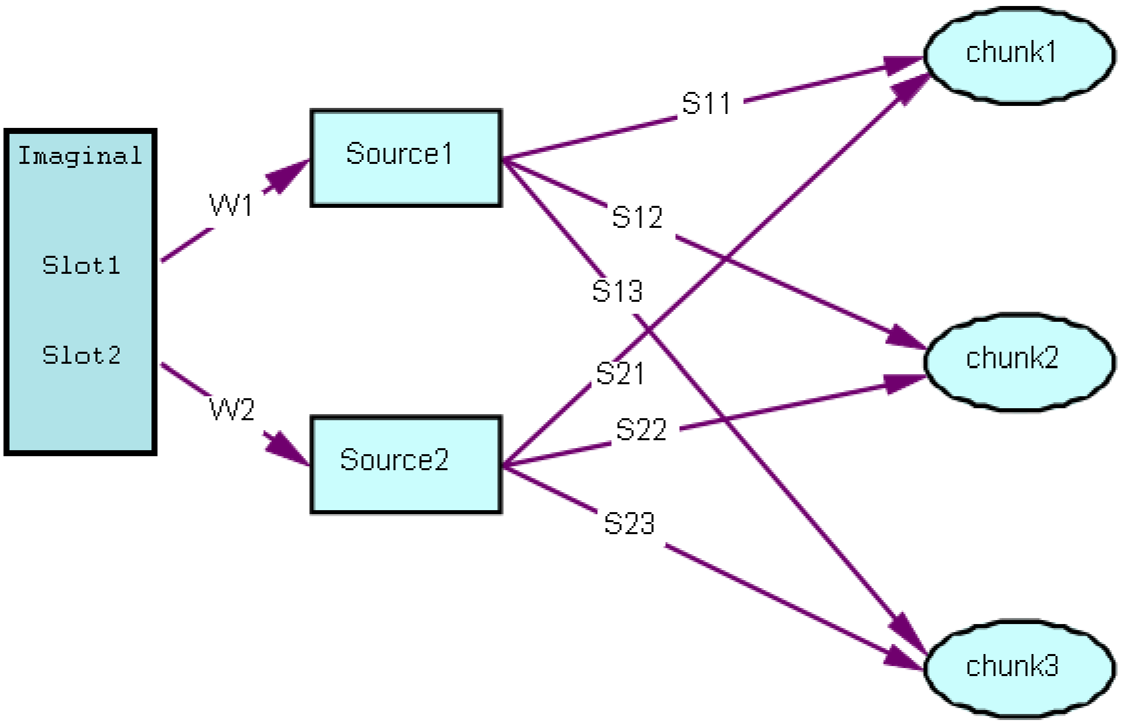
***fan$_j$*:** is the number of chunks in declarative memory in which *j* is the value of a slot plus one for chunk *j* being associated with itself. [That assumes the simple case where chunk *j* does not appear in more than one slot of any given chunk *i* which will be the case for the models in this unit. See the reference manual or the modeling text for this unit for the more general description.]

That is the general form of the spreading activation equation. However, by default, only the **imaginal** buffer serves as a source of activation. The ***W$_{imaginal}$*** value defaults to 1 (set with the :imaginal-activation parameter) and for all other buffers, ***W$_{buffer}$*,** defaults to 0, but can be set to non-zero values with corresponding buffer's spreading activation parameter. For the goal buffer that parameter is :ga and for all others it is :<*buffer*>-activation (where <*buffer*> is replaced with the actual name of the buffer for example :visual-activation for the **visual** buffer). Therefore, in the default case, the activation equation can be simplified to:

$$A_i = B_i + \sum_j W_j S_{ji} + \varepsilon$$

With W reflecting the value of the :imaginal-activation parameter and Wj being W/n where n is the number of chunks in slots of the current **imaginal** buffer chunk.

Here is a diagram to help you visualize how the spreading activation works. Consider an imaginal chunk which has two chunks in its slots when a retrieval is requested and that there are three chunks in declarative memory which match the retrieval request for which the activations need to be determined.



Each of the potential chunks also has a base-level activation which we will denote as B_i, and thus the total activation of the three chunks are:

$$A_1 = B_1 + W_1 S_{11} + W_2 S_{21}$$
$$A_2 = B_2 + W_1 S_{12} + W_2 S_{22}$$
$$A_3 = B_3 + W_1 S_{13} + W_2 S_{23}$$

and with the default value of 1.0 for the imaginal activation W1 = W2 = 1/2.

There are two notes about using spreading activation. First, by default, spreading activation is disabled because :mas defaults to the value **nil**. In order to enable the spreading activation calculation :mas must be set to a positive value. The other thing to note is that there is no recommended value for the :mas parameter, but one almost always wants to set :mas high enough that all of the S_{ji} values are positive.

5.2 The Fan Effect

Anderson (1974) performed an experiment in which participants studied 26 facts such as the following sentences:

```
1. A hippie is in the park.
2. A hippie is in the church.
3. A hippie is in the bank.
```

```
  4. A captain is in the park.
  5. A captain is in the cave.
  6. A debutante is in the bank.
  7. A fireman is in the park.
  8. A giant is in the beach.
  9. A giant is in the dungeon.
 10. A giant is in the castle.
 11. A earl is in the castle.
 12. A earl is in the forest.
 13. A lawyer is in the store.
 ...
```

After studying these facts, they had to judge whether they saw facts such as the following:

```
A hippie is in the park.
A hippie is in the cave.
A lawyer is in the store.
A lawyer is in the park.
A debutante is in the bank.
A debutante is in the cave.
A captain is in the bank.
```

which contained both studied sentences (targets) and new sentences (foils).

The people and locations for the study sentences could occur in any of one, two, or three of the study sentences. That is called their fan. The following tables show the recognition latencies from the experiment in seconds for targets and foils as a function of person fan and location fan:

	Targets				Foils			
Location	Person Fan				Person Fan			
Fan	1	2	3	Mean	1	2	3	Mean
1	1.111	1.174	1.222	1.169	1.197	1.221	1.264	1.227
2	1.167	1.198	1.222	1.196	1.250	1.356	1.291	1.299
3	1.153	1.233	1.357	1.248	1.262	1.471	1.465	1.399
Mean	1.144	1.202	1.357	1.20	1.236	1.349	1.340	1.308

The main effects in the data are that as the fan increases the time to respond increases and that foil sentences take longer to respond to than the targets. We will now show how these effects can be modeled using spreading activation.

5.3 Fan Effect Model

The **fan** model in the unit 5 materials contains a model for the testing phase of the experiment. The study portion of the task is not included for simplicity and the model already has chunks in declarative memory that encode all of the studied sentences. It can perform one trial of the testing phase when run. Here is a trace of the model performing one trial for the target sentence "The lawyer is in the store":

```
> (fan-sentence-model "lawyer" "store" t 'person)
0.000   VISION      SET-BUFFER-CHUNK VISUAL-LOCATION VISUAL-LOCATION0-0 REQUESTED NIL
0.000   PROCEDURAL  CONFLICT-RESOLUTION
0.050   PROCEDURAL  PRODUCTION-FIRED FIND-PERSON
0.050   PROCEDURAL  CLEAR-BUFFER IMAGINAL
0.050   PROCEDURAL  CLEAR-BUFFER VISUAL-LOCATION
0.050   VISION      Find-location
0.050   VISION      SET-BUFFER-CHUNK VISUAL-LOCATION VISUAL-LOCATION1-0
0.050   PROCEDURAL  CONFLICT-RESOLUTION
0.100   PROCEDURAL  PRODUCTION-FIRED ATTEND-VISUAL-LOCATION
0.100   PROCEDURAL  CLEAR-BUFFER VISUAL-LOCATION
0.100   PROCEDURAL  CLEAR-BUFFER VISUAL
0.100   PROCEDURAL  CONFLICT-RESOLUTION
0.185   VISION      Encoding-complete VISUAL-LOCATION1-0-0 NIL
0.185   VISION      SET-BUFFER-CHUNK VISUAL TEXT0
0.185   PROCEDURAL  CONFLICT-RESOLUTION
0.235   PROCEDURAL  PRODUCTION-FIRED RETRIEVE-MEANING
0.235   PROCEDURAL  CLEAR-BUFFER VISUAL
0.235   PROCEDURAL  CLEAR-BUFFER RETRIEVAL
0.235   DECLARATIVE START-RETRIEVAL
0.235   DECLARATIVE RETRIEVED-CHUNK LAWYER
0.235   DECLARATIVE SET-BUFFER-CHUNK RETRIEVAL LAWYER
0.235   PROCEDURAL  CONFLICT-RESOLUTION
0.250   IMAGINAL    SET-BUFFER-CHUNK IMAGINAL CHUNK0
0.250   PROCEDURAL  CONFLICT-RESOLUTION
0.300   PROCEDURAL  PRODUCTION-FIRED ENCODE-PERSON
0.300   PROCEDURAL  CLEAR-BUFFER RETRIEVAL
0.300   PROCEDURAL  CLEAR-BUFFER VISUAL-LOCATION
0.300   VISION      Find-location
0.300   VISION      SET-BUFFER-CHUNK VISUAL-LOCATION VISUAL-LOCATION5-0
0.300   PROCEDURAL  CONFLICT-RESOLUTION
0.350   PROCEDURAL  PRODUCTION-FIRED ATTEND-VISUAL-LOCATION
0.350   PROCEDURAL  CLEAR-BUFFER VISUAL-LOCATION
0.350   PROCEDURAL  CLEAR-BUFFER VISUAL
0.350   PROCEDURAL  CONFLICT-RESOLUTION
0.435   VISION      Encoding-complete VISUAL-LOCATION5-0-0 NIL
0.435   VISION      SET-BUFFER-CHUNK VISUAL TEXT1
0.435   PROCEDURAL  CONFLICT-RESOLUTION
0.485   PROCEDURAL  PRODUCTION-FIRED RETRIEVE-MEANING
0.485   PROCEDURAL  CLEAR-BUFFER VISUAL
0.485   PROCEDURAL  CLEAR-BUFFER RETRIEVAL
0.485   DECLARATIVE START-RETRIEVAL
0.485   DECLARATIVE RETRIEVED-CHUNK STORE
0.485   DECLARATIVE SET-BUFFER-CHUNK RETRIEVAL STORE
0.485   PROCEDURAL  CONFLICT-RESOLUTION
0.535   PROCEDURAL  PRODUCTION-FIRED ENCODE-LOCATION
0.535   PROCEDURAL  CLEAR-BUFFER RETRIEVAL
0.535   PROCEDURAL  CONFLICT-RESOLUTION
0.585   PROCEDURAL  PRODUCTION-FIRED RETRIEVE-FROM-PERSON
0.585   PROCEDURAL  CLEAR-BUFFER RETRIEVAL
0.585   DECLARATIVE START-RETRIEVAL
0.585   PROCEDURAL  CONFLICT-RESOLUTION
0.839   DECLARATIVE RETRIEVED-CHUNK P13
0.839   DECLARATIVE SET-BUFFER-CHUNK RETRIEVAL P13
0.839   PROCEDURAL  CONFLICT-RESOLUTION
0.889   PROCEDURAL  PRODUCTION-FIRED YES
0.889   PROCEDURAL  CLEAR-BUFFER IMAGINAL
0.889   PROCEDURAL  CLEAR-BUFFER RETRIEVAL
```

```
0.889   PROCEDURAL  CLEAR-BUFFER MANUAL
0.889   MOTOR       PRESS-KEY KEY k
0.889   PROCEDURAL  CONFLICT-RESOLUTION
1.039   PROCEDURAL  CONFLICT-RESOLUTION
1.089   PROCEDURAL  CONFLICT-RESOLUTION
1.099   MOTOR       OUTPUT-KEY #(8 4)
1.099   PROCEDURAL  CONFLICT-RESOLUTION
1.189   PROCEDURAL  CONFLICT-RESOLUTION
1.189   ------      Stopped because no events left to process
```

To run the model through one trial of the test phase you can call the function **fan-sentence-model**. It takes four parameters. The first is a string of the person for the probe sentence. The second is a string of the location for the probe sentence. The third is whether the correct answer is true (t) or false (nil), and the last is either ‘person or ‘location to choose which of the retrieval productions is used (more on that later). The model can be run over each of the conditions to produce a data fit using the **fan-experiment** function (you will probably want to set the :v parameter to **nil** before running the whole experiment):

```
> (fan-experiment)
CORRELATION:  0.864
MEAN DEVIATION:  0.053

TARGETS:
                         Person fan
  Location      1             2             3
    fan
     1       1.099 (T  )   1.157 (T  )   1.205 (T  )
     2       1.157 (T  )   1.227 (T  )   1.286 (T  )
     3       1.205 (T  )   1.286 (T  )   1.354 (T  )

FOILS:
     1       1.245 (T  )   1.290 (T  )   1.328 (T  )
     2       1.290 (T  )   1.335 (T  )   1.373 (T  )
     3       1.328 (T  )   1.373 (T  )   1.411 (T  )
```

Two parameters were estimated to produce that fit to the data. They are the latency factor, which is the *F* in the retrieval latency equation from last chapter, set to .63 and the maximum associative strength, the *S* parameter in the S_{ji} equation above, set to 1.6. The spreading activation value for the **imaginal** buffer is set to the default value of 1.0. We will now look at how this model performs the task and how spreading activation leads to the effects in the data.

5.3.1 Model Representations

The study sentences are encoded in chunks like this:

```
P13
   RELATION  IN
   ARG1  LAWYER
   ARG2  STORE
```

which are propositions encoding the result of past study in the form of an association among the concepts (in this case in, lawyer, and store for “The lawyer is in the store”).

There are also meaning chunks which connect the text read from the display to the concepts. For instance, relevant to the case above we have

```
LAWYER
   WORD  "lawyer"

STORE
   WORD  "store"
```

The base-level activations of these meaning chunks have been set to 10 to reflect the fact that they are well practiced and should not fail to be retrieved, but the activations of the comprehend-sentence chunks are left at the default of 0 to reflect that they are relatively newer having only been learned during this experiment.

5.3.2 Perceptual Encoding

In this section we will briefly describe the productions that perform the perceptual portion of the trial. This is similar to the steps that have been done in previous models and thus it should be fairly familiar. One small difference is that this model does not use explicit state markers in the goal (in fact it does not place a chunk into the **goal** buffer at all) and instead relies on the states of the buffers and modules involved to constrain the productions.

The entire sentence is presented on the screen, but the model only reads the person and location words from the display to perform the task. If the model were to read all of the words in the sentence it would be difficult to be able to respond fast enough to match the experimental data, and in fact studies of the fan effect done using an eye tracker verify that participants generally only fixate those two words from the sentences during the testing trials. To make this easier to handle for the model the sentences are presented with the words in fixed locations on the display. To read and encode the words the model goes through a four step process.

The first production to fire issues a request to the **visual-location** buffer to find the person word and it also requests that the imaginal module create a new chunk to hold the sentence being read from the screen:

```
(P find-person
   ?visual-location>
      buffer      unrequested
   ?imaginal>
      state       free

   ==>
   +imaginal>

   +visual-location>
      ISA         visual-location
    > screen-x    105
```

```
    < screen-x    135)
```

Although the text for the word always starts at the same location its exact position will vary based on the length of the word, and thus a range test is used to specify where that word should be found.

Also of interest in that production is the first query on the LHS. The check that the **visual-location** buffer holds a chunk which was not requested is a way to test that a new display has been presented. The buffer stuffing mechanism will automatically place a chunk into the buffer if it is empty when the screen changes and because that chunk was not the result of a request it is tagged as unrequested. Thus, this production will match whenever the screen has recently changed if the **visual-location** buffer was empty at the time of the change and the **imaginal** module is not busy.

The next production harvests the requested visual-location and requests a shift of attention to it:

```
(P attend-visual-location
   =visual-location>

   ?visual-location>
      buffer      requested
   ?visual>
      state       free
   ==>
   +visual>
      cmd         move-attention
      screen-pos  =visual-location)
```

Then the chunk in the **visual** buffer is harvested and a retrieval request is made to request the chunk that represents the meaning of that word:

```
(P retrieve-meaning
   =visual>
      ISA         visual-object
      value       =word
   ==>
   +retrieval>
      ISA         meaning
      word        =word)
```

Finally, that retrieval request is harvested and the meaning chunk is placed into a slot of the chunk in the **imaginal** buffer:

```
(P encode-person
   =retrieval>
```

```
   =imaginal>
      ISA         comprehend-sentence
      arg1        nil
   ==>
   =imaginal>
      arg1        =retrieval
   +visual-location>
      ISA         visual-location
    > screen-x    400
    < screen-x    430)
```

This production then issues the visual-location request to find the location word and the same sequence of productions fire to attend and encode the location ending with the encode-location production firing instead of encode-person.

5.3.3 Determining the Response

After the encoding has happened the imaginal chunk will look like this for the sentence "The lawyer is in the store.":

```
IMAGINAL: CHUNK0-0
CHUNK0-0
   ARG1  LAWYER
   ARG2  STORE
```

At that point one of these two productions will be selected and fired to retrieve a study sentence:

```
(P retrieve-from-person
   =imaginal>
      ISA         comprehend-sentence
      arg1        =person
      arg2        =location
   ?retrieval>
      state       free
      buffer      empty
   ==>
   =imaginal>
   +retrieval>
      ISA         comprehend-sentence
      arg1        =person)


(P retrieve-from-location
   =imaginal>
      ISA         comprehend-sentence
```

```
      arg1         =person
      arg2         =location
   ?retrieval>
      state        free
      buffer       empty
   ==>
   =imaginal>
   +retrieval>
      ISA          comprehend-sentence
      arg2         =location)
```

A thorough model of the task would have those two productions competing and one would randomly be selected. However, to simplify things for demonstration the experiment code which runs this task forces one or the other to be selected for each trial. The data is then averaged over two runs of each trial with one trial using retrieve-from-person and the other using retrieve-from-location.

One important thing to notice is that those productions request the retrieval of a studied chunk based only on one of the items from the probe sentence. By doing so it ensures that one of the study sentences will be retrieved instead of a complete failure in the event of a foil. If retrieval failure were used by the model to detect the foils then there would be no difference in response times for the foil probes because the time to fail is based solely upon the retrieval threshold. However, the data clearly shows that the fan of the items affects the time to respond to both targets and foils.

After one of those productions fires a chunk representing a study trial will be retrieved and one of the following productions will fire to produce a response:

```
(P yes
   =imaginal>
      ISA          comprehend-sentence
      arg1         =person
      arg2         =location
   =retrieval>
      ISA          comprehend-sentence
      arg1         =person
      arg2         =location
   ?manual>
      state        free
   ==>
   +manual>
      cmd          press-key
      key          "k")

(P mismatch-person
   =imaginal>
      ISA          comprehend-sentence
```

```
      arg1         =person
      arg2         =location
   =retrieval>
      ISA          comprehend-sentence
   -  arg1         =person
   ?manual>
      state        free
   ==>
   +manual>
      ISA          press-key
      key          "d")

(P mismatch-location
   =imaginal>
      ISA          comprehend-sentence
      arg1         =person
      arg2         =location
   =retrieval>
      ISA          comprehend-sentence
    - arg2         =location
   ?manual>
      state        free
   ==>
   +manual>
      cmd          press-key
      key          "d")
```

If the retrieved sentence matches the probe then the model responds with the true response, “k”, and if either one of the components does not match then the model responds with “d”.

5.4 Analyzing the Retrieval of the Critical Study Chunk in the Fan model

The perceptual and encoding actions the model performs for this task have a fixed cost of .585 seconds and the time to respond after retrieving a comprehend-sentence chunk is .260 seconds. Those times are constant across all trials. The difference in the conditions will result from the time it takes to retrieve the studied sentence. Recall from the last unit that the time to retrieve a chunk *i* is based on its activation and specified by the equation:

$$Time_i = Fe^{-A_i}$$

Thus, it is differences in the activations of the chunks representing the studied items which will result in the different times to respond to different trials.

The chunk in the **imaginal** buffer at the time of the retrieval (after either retrieve-from-person or retrieve-from-location fires) will look like this:

```
IMAGINAL: CHUNK0-0
CHUNK0-0
   ARG1  person
   ARG2  location
```

where *person* and *location* will be the chunks that represent the meanings for the particular probe being presented.

The retrieval request will look like this:

```
+retrieval>
    arg1 person
```

or this:

```
+retrieval>
    arg2 location
```

depending on which of the productions was chosen to perform the retrieval.

The important thing to note is that because the sources of activation in the buffer are the same for either retrieval request the spreading activation will not differ between the two cases. You might wonder then why we would need to have both options. That will be described in the detailed examples below.

5.4.1 A note on chunks in buffers and the :dcnn parameter

Something that has been mentioned before is that buffers hold copies of chunks. A side effect of that is that when the name of that chunk is used (as is done with the retrievals in the encode-person and encode-location productions) it does not match the name of the original chunk. Thus, the chunk in the **imaginal** buffer will look like this after the encode-person production fires and the modification to the **imaginal** buffer occurs:

```
CHUNK0-0
   ARG1  LAWYER-0
```

because the copy of the lawyer chunk, named laywer-0, which was in the **retrieval** buffer was placed into the arg1 slot.

The lawyer-0 chunk is not modified by the model while it is in the **retrieval** buffer, and the **retrieval** buffer is also cleared by that production. Thus, once that clearing occurs the laywer-0 chunk is merged into declarative memory and because it is a perfect match to the lawyer chunk those two chunks are merged together so that the names lawyer-0 and lawyer refer to the same chunk. Both names may occur however in the slots of other chunks.

As discussed in the previous unit, the :ncnar parameter can be used to have the system normalize such references to make it easier to debug the system. If :ncnar is enabled (not **nil**) then the setting of the :dcnn (dynamic chunk name normalizing) parameter determines when those names are corrected. If :dcnn is set to **t**, which is the default value, then those changes are made immediately while the model runs. Since this model leaves the :ncnar and :dcnn parameters at their default values of **t**, the chunk in the **imaginal** buffer will be changed to look like this after the lawyer-0 and lawyer chunks are merged:

```
CHUNK0-0
   ARG1  LAWYER
```

If :dcnn were set to **nil**, then the arg1 slot would continue to hold the value lawyer-0 until the model stopped running at which time the chunk0-0 chunk would be updated if the :ncnar parameter were not **nil**. The important thing to note is that regardless of which name is shown in the slot, once those chunks are merged the same chunk is being referenced whether a particular slot value is normalized or not.

Often having :dcnn and :ncnar set to **t** makes it easier to debug a model, but sometimes it may be useful to disable the dynamic updating so that one can more directly track a reference to a chunk from a buffer, even if it is eventually merged, instead of having that reference lost. For this unit the demonstration models leave :dcnn and :ncnar set to their default values of **t**, thus the slot values are dynamically adjusted as the model runs.

5.4.2 A simple target trial

The first case we will look at is the target sentence “The lawyer is in the store”. Both the person and location in this sentence have a fan of one in the experiment – they each only occur in that one study sentence.

The **imaginal** buffer’s chunk looks like this at the time of the critical retrieval (as discussed above):

```
CHUNK0-0
   ARG1  LAWYER
   ARG2  STORE
```

We will now look at the retrieval which results from the retrieve-from-person production firing. For the following traces we have enabled the activation trace parameter (:act) by setting it to t. That causes additional information to be displayed in the trace when a retrieval attempt is made. It shows all of the chunks that were attempted to be matched, and then for each that does match it shows all the details of the activation computation. Here is the trace of the model when that retrieval occurs:

```
     0.585   DECLARATIVE            START-RETRIEVAL
Chunk P13 matches
Chunk P12 does not match
Chunk P11 does not match
Chunk P10 does not match
Chunk P9 does not match
```

```
Chunk P8 does not match
Chunk P7 does not match
Chunk P6 does not match
Chunk P5 does not match
Chunk P4 does not match
Chunk P3 does not match
Chunk P2 does not match
Chunk P1 does not match
Computing activation for chunk P13
Computing base-level
Starting with blc: 0.0
Total base-level: 0.0
Computing activation spreading from buffers
  Spreading 1.0 from buffer IMAGINAL chunk CHUNK0-0
    sources of activation are: (STORE LAWYER)
    Spreading activation  0.45342642 from source STORE level 0.5 times Sji 0.90685284
    Spreading activation  0.45342642 from source LAWYER level 0.5 times Sji 0.90685284
Total spreading activation: 0.90685284
Adding transient noise 0.0
Adding permanent noise 0.0
Chunk P13 has an activation of: 0.90685284
Chunk P13 has the current best activation 0.90685284
Chunk P13 with activation 0.90685284 is the best
     0.585   PROCEDURAL             CONFLICT-RESOLUTION
     0.839   DECLARATIVE            RETRIEVED-CHUNK P13
```

In this case, the only chunk which matches the request is chunk p13. Note that this would look exactly the same if the retrieve-from-location production had fired because it would still be the only chunk that matched the request and the sources of activation are the same regardless of which one fires.

Remember that we have set the parameter F to .63, the parameter S to 1.6, and the base-level activation for the comprehend-sentence chunks is 0 in this model.

Looking at this trace, we see the S_{ji} values from store to p13 and lawyer to p13 are both approximately .907. That comes from the equation:

$$S_{ji} = S - \ln(fan_j)$$

The value of S was estimated to fit the data as 1.6 and the chunk fan of both the store and lawyer chunks is 2 (not the same as the fan from the experiment which is only one) because they each occur as a slot value in only the p13 chunk plus each chunk is always credited with a reference to itself. Then substituting into the equation we get:

$$S_{(store)(p13)} = S_{(lawyer)(p13)} = 1.6 - \ln(2) = 0.90685284$$

The W_j values (called the level in the activation trace) are .5 because the source activation from the **imaginal** buffer is the 1.0 value which was set and there are two source chunks.

Thus the activation of chunk p13 is:

$$A_i = B_i + \sum_j W_j S_{ji}$$
$$A_{p13} = 0 + (.5*.907) + (.5*.907) = .907$$

Finally, we see the time to complete the retrieval (the time between the start-retrieval and the retrieved-chunk actions) is .254 seconds (.839- .585) and that was computed as:

$$Time_i = Fe^{-A_i}$$
$$Time_{p13} = .63e^{-.907} = 0.25435218$$

Adding that retrieval time to the fixed costs of .585 seconds to do the perception and encoding and the .26 seconds to perform the response gives us a total of 1.099 seconds, which is the value in the fan 1-1 cell of the model data for targets presented above.

Now that we have looked at the details of how the retrieval and total response times are determined for the simple case we will look at a few other cases.

5.4.3 A different target trial

The target sentence “The hippie is in the bank” is a more interesting case to look at. Hippie is the person in three of the study sentences and bank is the location in two of them. Now we will see why it takes the model longer to respond to such a probe. Here are the critical components from the trace when retrieve-from-person is chosen:

```
Computing activation for chunk P3
Computing base-level
Starting with blc: 0.0
Total base-level: 0.0
Computing activation spreading from buffers
  Spreading 1.0 from buffer IMAGINAL chunk CHUNK0-0
    sources of activation are: (BANK HIPPIE)
    Spreading activation  0.25069386 from source BANK level 0.5 times Sji 0.5013877
    Spreading activation  0.10685283 from source HIPPIE level 0.5 times Sji 0.21370566
Total spreading activation: 0.3575467
Adding transient noise 0.0
Adding permanent noise 0.0
Chunk P3 has an activation of: 0.3575467
Chunk P3 has the current best activation 0.3575467
Computing activation for chunk P2
Computing base-level
Starting with blc: 0.0
Total base-level: 0.0
Computing activation spreading from buffers
  Spreading 1.0 from buffer IMAGINAL chunk CHUNK0-0
    sources of activation are: (BANK HIPPIE)
    Spreading activation  0.0 from source BANK level 0.5 times Sji 0.0
    Spreading activation  0.10685283 from source HIPPIE level 0.5 times Sji 0.21370566
Total spreading activation: 0.10685283
Adding transient noise 0.0
Adding permanent noise 0.0
Chunk P2 has an activation of: 0.10685283

Computing activation for chunk P1
```

```
Computing base-level
Starting with blc: 0.0
Total base-level: 0.0
Computing activation spreading from buffers
  Spreading 1.0 from buffer IMAGINAL chunk CHUNK0-0
    sources of activation are: (BANK HIPPIE)
    Spreading activation  0.0 from source BANK level 0.5 times Sji 0.0
    Spreading activation  0.10685283 from source HIPPIE level 0.5 times Sji 0.21370566
Total spreading activation: 0.10685283
Adding transient noise 0.0
Adding permanent noise 0.0
Chunk P1 has an activation of: 0.10685283

Chunk P3 with activation 0.3575467 is the best
```

There are three chunks that match the request for a chunk with an arg1 value of hippie. Each receives the same amount of activation being spread from hippie. Because hippie is a member of three chunks it has a chunk fan of 4 and thus the $S_{(hippie)i}$ value is:

$$S_{(hippie)i} = 1.6 - \ln(4) = 0.21370566$$

Chunk p3 also contains the chunk bank in its arg2 slot and thus receives the source spreading from it as well.

Now we will look at the case when retrieve-from-location fires for this probe sentence:

```
Computing activation for chunk P6
Computing base-level
Starting with blc: 0.0
Total base-level: 0.0
Computing activation spreading from buffers
  Spreading 1.0 from buffer IMAGINAL chunk CHUNK0-0
    sources of activation are: (BANK HIPPIE)
    Spreading activation  0.25069386 from source BANK level 0.5 times Sji 0.5013877
    Spreading activation  0.0 from source HIPPIE level 0.5 times Sji 0.0
Total spreading activation: 0.25069386
Adding transient noise 0.0
Adding permanent noise 0.0
Chunk P6 has an activation of: 0.25069386
Chunk P6 has the current best activation 0.25069386

Computing activation for chunk P3
Computing base-level
Starting with blc: 0.0
Total base-level: 0.0
Computing activation spreading from buffers
  Spreading 1.0 from buffer IMAGINAL chunk CHUNK0-0
    sources of activation are: (BANK HIPPIE)
    Spreading activation  0.25069386 from source BANK level 0.5 times Sji 0.5013877
    Spreading activation  0.10685283 from source HIPPIE level 0.5 times Sji 0.21370566
Total spreading activation: 0.3575467
Adding transient noise 0.0
Adding permanent noise 0.0
Chunk P3 has an activation of: 0.3575467
Chunk P3 is now the current best with activation 0.3575467

Chunk P3 with activation 0.3575467 is the best
```

In this case there are only two chunks which match the request for a chunk with an arg2 value of bank.

Regardless of which production fired to request the retrieval, chunk p3 had the highest activation because it received spreading activation from both sources. Thus, even if there is more than one chunk which matches the retrieval request issued by retrieve-from-person or retrieve-from-location the correct study sentence will always be retrieved because its activation will be the highest, and that activation value will be the same in both cases.

Notice that the activation of chunk p3 is less than the activation that chunk p13 had in the previous example because the source activation being spread to p3 is less. That is because the sources in that case have a higher fan, and thus a lesser S_{ji}. Because the activation is smaller, it takes longer to retrieve such a fact and that gives us the difference in response time effect of fan in the data.

5.4.4 A foil trial

Now we will look at a foil trial. The foil probe "The giant is in the bank" is similar to the target that we looked at in the last section. The person has an experimental fan of three and the location has an experimental fan of two. This time however there is no matching study sentence. Here are the critical components from the trace when retrieve-from-person is chosen:

```
Computing activation for chunk P10
Computing base-level
Starting with blc: 0.0
Total base-level: 0.0
Computing activation spreading from buffers
  Spreading 1.0 from buffer IMAGINAL chunk CHUNK0-0
    sources of activation are: (BANK GIANT)
    Spreading activation  0.0 from source BANK level  0.5 times Sji 0.0
    Spreading activation  0.10685283 from source GIANT level  0.5 times Sji 0.21370566
Total spreading activation: 0.10685283
Adding transient noise 0.0
Adding permanent noise 0.0
Chunk P10 has an activation of: 0.10685283
Chunk P10 has the current best activation 0.10685283
Computing activation for chunk P9
Computing base-level
Starting with blc: 0.0
Total base-level: 0.0
Computing activation spreading from buffers
  Spreading 1.0 from buffer IMAGINAL chunk CHUNK0-0
    sources of activation are: (BANK GIANT)
    Spreading activation  0.0 from source BANK level  0.5 times Sji 0.0
    Spreading activation  0.10685283 from source GIANT level  0.5 times Sji 0.21370566
Total spreading activation: 0.10685283
Adding transient noise 0.0
Adding permanent noise 0.0
Chunk P9 has an activation of: 0.10685283
Chunk P9 matches the current best activation 0.10685283

Computing activation for chunk P8
Computing base-level
```

```
Starting with blc: 0.0
Total base-level: 0.0
Computing activation spreading from buffers
  Spreading 1.0 from buffer IMAGINAL chunk CHUNK0-0
    sources of activation are: (BANK GIANT)
    Spreading activation  0.0 from source BANK level  0.5 times Sji 0.0
    Spreading activation  0.10685283 from source GIANT level  0.5 times Sji 0.21370566
Total spreading activation: 0.10685283
Adding transient noise 0.0
Adding permanent noise 0.0
Chunk P8 has an activation of: 0.10685283
Chunk P8 matches the current best activation 0.10685283

Chunk P10 chosen among the chunks with activation 0.10685283
```

There are three chunks that match the request for a chunk with an arg1 value of giant and each receives the same amount of activation being spread from giant. However, none contain an arg2 value of bank. Thus they only get activation spread from one source and have a lesser activation value than the corresponding target sentence probe had. Because the activation is smaller, the retrieval time is greater. This results in the effect of foil trials taking longer than target trials.

Before concluding this section however, let us look at the trace if retrieve-from-location were to fire for this foil:

```
Computing activation for chunk P6
Computing base-level
Starting with blc: 0.0
Total base-level: 0.0
Computing activation spreading from buffers
  Spreading 1.0 from buffer IMAGINAL chunk CHUNK0-0
    sources of activation are: (BANK GIANT)
    Spreading activation  0.25069386 from source BANK level  0.5 times Sji 0.5013877
    Spreading activation  0.0 from source GIANT level  0.5 times Sji 0.0
Total spreading activation: 0.25069386
Adding transient noise 0.0
Adding permanent noise 0.0
Chunk P6 has an activation of: 0.25069386
Chunk P6 has the current best activation 0.25069386

Computing activation for chunk P3
Computing base-level
Starting with blc: 0.0
Total base-level: 0.0
Computing activation spreading from buffers
  Spreading 1.0 from buffer IMAGINAL chunk CHUNK0-0
    sources of activation are: (BANK GIANT)
    Spreading activation  0.25069386 from source BANK level  0.5 times Sji 0.5013877
    Spreading activation  0.0 from source GIANT level  0.5 times Sji 0.0
Total spreading activation: 0.25069386
Adding transient noise 0.0
Adding permanent noise 0.0
Chunk P3 has an activation of: 0.25069386
Chunk P3 matches the current best activation 0.25069386

Chunk P3 chosen among the chunks with activation 0.25069386
```

In this case there are only two chunks which match the request for a chunk with an arg2 value of bank. Again, the activation of the chunk retrieved is less than the corresponding target trial, but it is not the same as when retrieve-from-person fired. That is why the model is run with each of those productions fired once for each probe with the results being averaged together. Otherwise the foil data would only show the effect of fan for the item that was used to retrieve the study chunk.

5.5 Partial Matching

Up to now models have either always retrieved a chunk which matched the retrieval request or resulted in a failure to retrieve anything. Now we will look at modeling errors in recall in more detail. There are two kinds of errors that can occur. One is an error of commission when the wrong thing is recalled. This will occur when the activation of the wrong chunk is greater than the activation of the correct chunk. The second is an error of omission when nothing is recalled. This will occur when no chunk has activation above the retrieval threshold.

We will continue to look at productions from the fan model for now. In particular, this production requests the retrieval of a chunk:

```
(P retrieve-from-person
     =imaginal>
        ISA         comprehend-sentence
        arg1        =person
        arg2        =location
     ?retrieval>
        state       free
        buffer      empty
     ==>
     =imaginal>
     +retrieval>
        ISA         comprehend-sentence
        arg1        =person)
```

In this case an attempt is being made to retrieve a chunk with a particular person (the value bound to =person) that had been studied. If =person were the chunk giant, this retrieval request would be looking for a chunk of the form:

```
+retrieval>
     isa comprehend-sentence
     arg1 giant
```

As was shown above, there were three chunks which matched that request in the study set and one of those will be retrieved.

However, let us consider the case where there had been no study sentences with the person giant but there had been a sentence with the person titan in the location being probed with giant i.e.

there was a study sentence "The titan is in the bank" and the test sentence is now "The giant is in the bank". In this situation one might expect that some human participants might incorrectly classify the probe sentence as one that was studied because of the similarity between the words giant and titan. The current model however could not make such an error.

Producing errors like that requires the use of the partial matching mechanism. When partial matching is enabled (by setting the :mp parameter to a number) the similarity between the chunks in the retrieval request and the chunks in the slots of the chunks in declarative memory are taken into consideration. The chunk with the highest activation is still the one retrieved, but with partial matching enabled that chunk might not have the exact slot values as specified in the retrieval request.

Adding the partial matching component into the activation equation, we now have the activation A_i of a chunk *i* defined fully as:

$$A_i = B_i + \sum_k \sum_j W_{kj} S_{ji} + \sum_l P M_{li} + \varepsilon$$

B_i, W_{kj}, S_{ji}, and ε have been discussed previously. The new term is the partial matching component.

Specification elements l**:** The matching summation is computed over the slot values of the retrieval specification.

Match Scale, *P*: This reflects the amount of weighting given to the similarity in slot *l*. This is a constant across all slots and is set with the :mp parameter (it is also often referred to as the mismatch penalty).

Match Similarities, M_{li}: The similarity between the value *l* in the retrieval specification and the value in the corresponding slot of chunk *i*.

The similarity value, the M_{li}, can be set by the modeler along with the scale on which they are defined. The scale range is set with a maximum similarity (set using the :ms parameter) and a maximum difference (set using the :md parameter). By default, :ms is 0 and :md is -1.0. The similarity between anything and itself is automatically set to the maximum similarity and by default the similarity between any other pair of values is the maximum difference. Note that maximum similarity defaults to 0 and similarity values are actually negative. If a slot value matches the request then it does not penalize the activation, but if it mismatches then the activation is decreased. To demonstrate partial matching in use we will look at two example models.

5.6 Grouped Recall

To first of these models is the model called **grouped**. This is a demonstration model of a grouped recall task which is based on a larger model of a complex recall experiment. As with the **fan** model, the studied items are already specified in the model, so it does not model the encoding and study of the items. In addition, the response times and error profiles of this model are not fit to any data. This demonstration model is designed to show the mechanism of partial matching and how it can lead to errors of commission and errors of omission. Because the model is not fit to any data, and the mechanism being studied does not rely on any of the perceptual or motor modules of ACT-R, they are not being used, and instead only a chunk in the **goal** buffer is used to hold both the task state and problem representation. This technique of using only the cognitive system in ACT-R can be useful when modeling a task where the timing is not important or other situations where accounting for a “real world” interaction is not necessary to accomplish the objectives of the model. The experiment description text for this unit gives the details of how that is accomplished in this model and in an alternate version of the **fan** model which also does not use the perceptual and motor modules.

If you check the parameter settings for this model you will see that it has a value of .15 for the transient noise *s* parameter and a retrieval threshold of -.5. Also, to simplify the demonstration, the spreading activation described above is disabled by not providing a value for the :mas parameter. This model is set up to recall a list of nine items which are encoded in groups of three elements. The list that should be recalled is (123) (456) (789). To run the model, call the run-grouped-recall function. That will print out the trace of the model doing the task and return a list of the model’s responses. Because the :seed parameter is set in the model you will always get the same run (you can remove the setting of the :seed parameter to produce different results if you would like to explore the model further) and that will be of the model responding with:

```
("1" "2" "3" "4" "6" "5" "7" "8")
```

Where it mis-ordered the recall of the 5 and 6 and failed to recall the last item, 9.

5.6.1 Error of Commission

If one turns on the activation trace for this model you will again see the details of the activation computations taking place. The following is from the activation trace of the error of commission when the model recalls 6 in the second position of the second group instead of the correct item, 5. The critical comparison is between item5, which should be retrieved and item6, which is instead retrieved:

```
Computing activation for chunk ITEM5
Computing base-level
Starting with blc: 0.0
Total base-level: 0.0
Computing partial matching component
  comparing slot GROUP
  Requested: = GROUP2  Chunk's slot value: GROUP2
  similarity: 0.0
  effective similarity value is 0.0
  comparing slot POSITION
  Requested: = SECOND  Chunk's slot value: SECOND
  similarity: 0.0
```

```
  effective similarity value is 0.0
Total similarity score 0.0
Adding transient noise -0.59634924
Adding permanent noise 0.0
Chunk ITEM5 has an activation of: -0.59634924
Chunk ITEM5 has the current best activation -0.59634924

Computing activation for chunk ITEM6
Computing base-level
Starting with blc: 0.0
Total base-level: 0.0
Computing partial matching component
  comparing slot GROUP
  Requested: = GROUP2  Chunk's slot value: GROUP2
  similarity: 0.0
  effective similarity value is 0.0
  comparing slot POSITION
  Requested: = SECOND  Chunk's slot value: THIRD
  similarity: -0.5
  effective similarity value is -0.5
Total similarity score -0.5
Adding transient noise 0.11740411
Adding permanent noise 0.0
Chunk ITEM6 has an activation of: -0.3825959
Chunk ITEM6 is now the current best with activation -0.3825959
...
Chunk ITEM6 with activation -0.3825959 is the best
```

In these examples the base-level activations, B_i, have their default value of 0, the match scale, *P*, has the value 1, and the only noise value is the transient component with an *s* of 0.15. So the calculations are really just a matter of adding up the match similarities, M_{li} and adding the transient noise.

One thing to notice is that the :recently-retrieved request parameter is specified in the request:

```
+retrieval>
   isa      item
   group   =group
   position second
   :recently-retrieved nil
```

Thus, only those chunks without a declarative finst are attempted for the matching. :recently-retrieved is not a slot of the chunk and thus does not undergo the partial matching calculation.

Looking at the matching of item5 above we see that it matches on both the group and position slots resulting in the addition of 0 to the base-level activation as a result of mismatch. It then receives an addition of about -0.596 in noise which is then its final activation value.

Next comes the matching of item6. The group slot matches the requested value of group2, but the position slots do not match. The requested value is second but item6 has a value of third. The similarity between second and third is set to -0.5 in the model, and that value is added to the activation. Then a transient noise of .117 is added to the activation for a total activation of -.383. This value is greater than the activation of item5 and thus because of random fluctuations item6 gets retrieved in error.

The similarities between the different positions are defined in the model using the set-similarities command:

```
(set-similarities
 (first second -0.5)
 (second third -0.5))
```

Similarity values are symmetric, thus it is not necessary to also specify (second first -0.5). The similarity between a chunk and itself has the value of maximum similarity therefore it is not necessary to specify (first first 0), and so on, for all the positions. Also, by default, different chunks have the maximum difference thus the similarity between first and third is -1 even though it is not specified.

5.6.2 Error of Omission

Here is the portion of the detailed trace relevant to the failure to recall the ninth item:

```
Computing activation for chunk ITEM9
Computing base-level
Starting with blc: 0.0
Total base-level: 0.0
Computing partial matching component
  comparing slot GROUP
  Requested: = GROUP3  Chunk's slot value: GROUP3
  similarity: 0.0
  effective similarity value is 0.0
  comparing slot POSITION
  Requested: = THIRD  Chunk's slot value: THIRD
  similarity: 0.0
  effective similarity value is 0.0
Total similarity score 0.0
Adding transient noise -0.5353896
Adding permanent noise 0.0
Chunk ITEM9 has an activation of: -0.5353896
Chunk ITEM9 has the current best activation -0.5353896
No chunk above the retrieval threshold: -0.5
```

We see that item9 starts out with an activation of 0 because it matches perfectly with the request and thus receives no penalty. However, it gets a transient noise of -.535 added to it which pushes its activation below the retrieval threshold and thus it cannot be retrieved. Because it is the only item chunk which is not marked as recently-retrieved it is the only one that can potentially be retrieved. Thus there are no chunks above the threshold and a retrieval failure occurs.

5.7 Simple Addition

The other model for the unit which uses partial matching is fit to experimental data. The task is an experiment performed by Siegler and Shrager on the relative frequencies of different responses by 4 year olds to addition problems. The children were asked to recall the answers to simple addition problems without counting on their fingers or otherwise computing the answer. It seems likely that many of the kids did not know the answers to the larger problems that were tested. So we will only focus on the addition table from 1+1 to 3+3, and here are the data they reported:

	0	1	2	3	4	5	6	7	8	Other (includes no response)
1+1	-	.05	.86	-	.02	-	.02	-	-	.06
1+2	-	.04	.07	.75	.04	-	.02	-	-	.09
1+3	-	.02	-	.10	.75	.05	.01	.03	-	.06
2+2	.02	-	.04	.05	.80	.04	-	.05	-	-
2+3	-	-	.07	.09	.25	.45	.08	.01	.01	.06
3+3	.04	-	-	.05	.21	.09	.48	-	.02	.11

The **siegler** model contains the functions to perform a version of this task with a model. As with the **grouped** model, there is no interface generated for the task, and thus it is not possible to run yourself through the experiment. That should not be too much of a problem however because one would guess that you would make very few errors if presented with such a task.

The function called **test-fact** will present 1 trial to the model and it requires the two numeric addends as its parameters. Thus, to run a trial of 1+2 you would call (test-fact 1 2). The model is reset before each trial of the task, and then run with those numbers being presented aurally. If the model responds by speaking a number, then that spoken text will be returned by **test-fact**. If the model does not respond then **test-fact** will return a value of **nil**. The function **run-siegler** takes one parameter which is the number of times to present each problem. It will then tally all of the responses, report the fit to the data, and display the results in a table like this:

```
> (run-siegler 500)
CORRELATION:  0.966
MEAN DEVIATION:  0.054
       0     1     2     3     4     5     6     7     8   Other
1+1  0.01  0.12  0.74  0.11  0.01  0.00  0.00  0.00  0.00  0.02
1+2  0.00  0.00  0.14  0.70  0.12  0.00  0.00  0.00  0.00  0.03
1+3  0.00  0.00  0.00  0.15  0.76  0.04  0.00  0.00  0.00  0.04
2+2  0.00  0.00  0.01  0.13  0.79  0.03  0.00  0.00  0.00  0.04
2+3  0.00  0.00  0.00  0.02  0.25  0.51  0.07  0.01  0.00  0.14
3+3  0.00  0.00  0.00  0.00  0.01  0.09  0.64  0.07  0.00  0.18
```

Like the **fan** model, this model does not rely on the **goal** buffer at all for tracking its progress. For this task the numbers are presented aurally to the model and it speaks its answer if it has one. The model builds up its representation of the problem in the **imaginal** buffer and relies on the module states and buffer contents to determine what needs to be done next.

Most of the conditions and actions in the productions for this model are similar to those that have been used in other tutorial models up until now. Thus, you should be able to understand and follow the operation of the model without it being described in detail here. However, there are two productions which have actions that have not been seen before in the tutorial. We will describe those new actions and then look at the parameter settings used to match the data.

5.7.1 A Modification Request

This production in the siegler model has an action for the **imaginal** buffer which has not been discussed previously in the tutorial:

```
(p harvest-arg2
   =retrieval>
     isa           number
   =imaginal>
     isa           plus-fact
     addend2       nil
   ?imaginal>
     state         free
  ==>
   *imaginal>
     addend2       =retrieval)
```

An action for a buffer which begins with a * is called a modification request. It works similarly to a request which is specified with a + in that it is asking the buffer's module to perform some action which can vary from module to module, but the modification request differs in that it does not clear the buffer automatically in the process of making the request. Not every module supports modification requests, but both the goal and imaginal modules do and they both handle them in the same manner.

A modification request to the goal or imaginal module is a request for the module to modify the chunk that is in the buffer in the same way that a production would modify the chunk with an = action. With the goal module the only difference between an =goal action and a *goal action will be which module is credited with the action. Consider this production from the count model in unit1:

```
(p start
   =goal>
      ISA         count-from
      start       =num1
      count       nil
 ==>
   =goal>
      ISA         count-from
      count       =num1
   +retrieval>
      ISA         count-order
      first       =num1
)
```

The =goal action in that production results in this output in the trace:

```
     0.050   PROCEDURAL                 MOD-BUFFER-CHUNK GOAL
```

Indicating that the procedural module modified the chunk in the **goal** buffer. If instead the start production used a *goal action like this:

```
(p start
   =goal>
      ISA         count-from
      start       =num1
      count       nil
 ==>
   *goal>
      ISA         count-from
      count       =num1
   +retrieval>
      ISA         count-order
      first       =num1
)
```

Then the output would look like this:

```
     0.050   PROCEDURAL             MODULE-MOD-REQUEST GOAL
     0.050   GOAL                   MOD-BUFFER-CHUNK GOAL
```

It shows that the procedural module made a modification request to the **goal** buffer and then the goal module actual performed the modification to the chunk in the buffer. That may not seem like an important distinction, but if one is trying to compare a model's actions to human brain activity then knowing "where" an action occurred is important.

For the imaginal module there is another difference between the =imaginal and the *imaginal actions. As we saw previously a request to create a chunk in the **imaginal** buffer has a time cost of 200ms. The same cost applies to modifications made to the chunk in the **imaginal** buffer by the imaginal module, whereas the production makes the modification immediately. Therefore if that start production were instead using the **imaginal** buffer:

```
(p start
   =imaginal>
      ISA         count-from
      start       =num1
      count       nil
 ==>
   =imaginal>
      ISA         count-from
      count       =num1
   )
```

We would see this trace for the =imaginal action:

```
   0.050   PROCEDURAL             PRODUCTION-FIRED START
```

```
   0.050   PROCEDURAL             MOD-BUFFER-CHUNK IMAGINAL
```

However if the production used the *imaginal action (which would also require a test that the module is not busy as should be done for all requests):

```
(p start
   =imaginal>
      ISA         count-from
      start       =num1
      count       nil
   ?imaginal>
      state       free
 ==>
   *imaginal>
      ISA         count-from
      count       =num1
   )
```

Then we would see this sequence of events in the trace:

```
     0.050   PROCEDURAL             PRODUCTION-FIRED START
     0.050   PROCEDURAL             MODULE-MOD-REQUEST IMAGINAL
     0.250   IMAGINAL               MOD-BUFFER-CHUNK IMAGINAL
```

Which shows the 200ms cost before the imaginal module makes the modification to the chunk in the buffer.

The *imaginal action is the recommended way to make changes to the chunk in the **imaginal** buffer because it has the time cost for the imaginal module to make the change. However if one isn’t as concerned about timing or the imaginal cost is not important for the task being modeld then the =imaginal actions can be used for simplicitly as has been done up to this point in the tutorial.

5.7.2 An indirect request

This production in the siegler model has a **retrieval** buffer request which has not been discussed previously in the tutorial:

```
(P harvest-answer
   =retrieval>
      ISA         plus-fact
      sum         =number
   =imaginal>
      isa         plus-fact
   ?imaginal>
```

```
      state       free
  ==>
   *imaginal>
      sum         =number
   +retrieval>    =number)
```

This production harvests the chunk in the **retrieval** buffer and copies the value from the sum slot of that chunk into the sum slot of the **imaginal** buffer and also makes what is called an indirect request to the **retrieval** buffer to retrieve the chunk which is contained in that slot. That chunk must be retrieved so that the value in its name slot can be used to speak the response.

An indirect request can be made through any buffer by specifying a chunk or a variable bound to a chunk as the only component of the request. The actual request which is sent to the module in such a situation is constructed as if all of the slots and values of that chunk were specified explicitly. Thus, if in the production above =number were bound to the chunk eight from the model:

```
(eight ISA number value 8 name "eight")
```

Then that retrieval request would be equivalent to this:

```
   +retrieval>
      value  8
      name   "eight"
```

In fact, the module which receives the request will see it exactly like that – it has no access to the name of the chunk which was used to make the indirect request. Therefore, an indirect request will be handled by the module exactly the same way as a normal request.

In this case, since it is a retrieval request, it will undergo the same activation calculations and be subject to partial matching just like any other retrieval request. Thus an indirect request to the **retrieval** buffer is not guaranteed to put that chunk into the buffer. In this model, the correct chunk should always be retrieved because it will match on all of its slots and thus receive no penalty to its activation while all the other number chunks will receive twice the maximum difference penalty to their activation since they will mismatch on both slots and there are no similarities set in the model between numbers as used in the value slot or the strings used in the name slot.

On a related note, if one absolutely must place a specific chunk into a particular buffer there is an action which will do that within a production, but that is not a recommended practice, and thus is not going to be covered in the tutorial.

5.7.3 Parameters to be adjusted

To achieve that fit to the data we will be using partial matching and adjusting the base-level activations of the plus-facts for the model. We are not going to use spreading activation for this demonstration. However, if you would like to explore the effect it has on the model feel free to enable it and experiment with adjusting the source spread from the **imaginal** buffer which is holding the contextual information in this task.

The specific parameters that we will need to adjust for the model are those related to activation in general (the retrieval threshold, :rt, and the activation noise, :ans), those related to partial matching (the similarities among the number chunks used as the addends of the plus-facts and the match scale value, :mp), and the base-level activation values of the chunks.

That is potentially a lot of free parameters in the model. Treating them that way one could likely produce an extremely strong fit to the reported data. However, doing that is not very practical nor does it result in a model that is of much use for demonstrating anything other than the ability to fit 60 data points using more than 60 parameters.

In the following sections we will describe the effects that the particular parameters have on the model's performance and outline an approach which can be taken to arrive at the parameter settings in a model.

5.7.4 Initial model

The first thing to do for the model is make sure that it can do the task. In this case that is hear the numbers, attempt to retrieve an addition fact, and then speak the result. To do that, we will start without enabling the subsymbolic components of the system. Making sure the model works right with basic symbolic information is a good start for modeling complex tasks because once the subsymbolic components are enabled and more sources of randomness or indeterminate behavior are introduced it can be very difficult to find potential errors in the productions or basic logic of the model.

The assumptions for the model are that the children know the numbers from zero through nine and that they have encountered the addition facts for problems with addends from zero to five. Thus these will be the declarative memory elements with which the model will start. Along with that, we are assuming that the children are not going to use any complex problem solving to try to remember the answers and that if there is a failure to remember a fact after one try the model will just give-up and answer that it does not know. For a task of this nature where we are modeling the aggregate data, using a single idealized strategy for the model is often a reasonable approach, and has been how all the other models seen so far in the tutorial operate. In other circumstances, particularly when individual participant data is being model, the specific strategy used to perform the task may be important, and in those cases it may be necessary to include different strategies into the model to account for the data.

With the model working correctly in a purely symbolic fashion we should see it answering correctly on every trial and here are the results of the model in that case providing the starting point for the adjustments to be made:

```
> (run-siegler 100)
CORRELATION:  0.943
```

```
MEAN DEVIATION:  0.127
       0     1     2     3     4     5     6     7     8   Other
1+1  0.00  0.00  1.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00
1+2  0.00  0.00  0.00  1.00  0.00  0.00  0.00  0.00  0.00  0.00
1+3  0.00  0.00  0.00  0.00  1.00  0.00  0.00  0.00  0.00  0.00
2+2  0.00  0.00  0.00  0.00  1.00  0.00  0.00  0.00  0.00  0.00
2+3  0.00  0.00  0.00  0.00  0.00  1.00  0.00  0.00  0.00  0.00
3+3  0.00  0.00  0.00  0.00  0.00  0.00  1.00  0.00  0.00  0.00
```

5.7.5 Making errors

Now that we have a model which performs perfectly we need to consider how we want it to model the errors. For this task we have chosen to use partial matching to do that. Specifically, we want the model to retrieve an incorrect addition fact as it does the task and also to sometimes fail to retrieve a an addition fact at all (an important source of the "other" results for the model). What we do not want it to do is retrieve an incorrect number chunk or fail to retrieve one while encoding the audio input or producing the vocal output. The reason for that is because we are assuming that the children know their numbers and thus do not produce errors because they are failing to hear or speak correctly. That is important because we are not just looking to have the model fit the data but to actually have it do so in a manner which seems plausible for the task.

To make those errors through partial matching will require that the model occasionally retrieve the wrong chunk for a request which looks like this:

```
+retrieval>
  ISA          plus-fact
  addend1      =val1
  addend2      =val2
```

where the =val1 and =val2 variables are bound to number chunks, for example one and three. Thus, the items which need to be similar are those number chunks which are the cues used in the retrieval. So, that is where we will start in setting the parameters.

5.7.6 Setting similarities

The similarity settings between the number chunks will affect the distribution of incorrect retrievals. While this looks like a lot of free parameters to be fit, in practice that is just not reasonable. For a situation like this, where the chunks represent numbers, it is better to set the similarity between two numbers based on the numerical difference between them using a single formula to specify all of the similarities. There is a lot of research into how people rate the similarity of numbers and there are many equations which have been proposed to describe it. For this task, we are going to use a linear function of the difference between the numbers.

Also, to keep things simple we will use the default range of similarity values for the model, which are from 0.0 for most similar to -1.0 for most dissimilar. Since we are working with numbers from 0-9 an obvious choice for setting them seems to be:

$$Similarity(a,b) = -(0.1 * |a - b|)$$

To set those similarities, we need to use the set-similarities command. Because the similarities are symmetric we only need to set each pair of numbers once and we do not need to set the similarity between a chunk and itself because that defaults to the most similar value. We also note that since the model only has chunks for encoding the facts with addends from 0-5 we only need to set the similarities for the chunks which are relevant to the task. Thus, here are the initial similarity values set for the model:

```
(Set-similarities
  (zero one -0.1)  (one two -0.1)  (two three -0.1)(three four -0.1)(four five -0.1)
  (zero two -0.2)  (one three -0.2)(two four -0.2) (three five -0.2)
  (zero three -0.3)(one four -0.3) (two five -0.3)
  (zero four -0.4) (one five -0.4)
  (zero five -0.5))
```

In addition to the similarities, we will also need to set the match scale parameter for the model. Adjusting the match scale will determine how much the similarity values affect the activation of the chunks since it is used to multiply the similarity values. Because we have chosen a linear scale for our similarity values we will actually be able to just use the match scale parameter to handle all of our adjustments instead of needing to adjust the available range or the parameter we chose in our similarity equation.

The similarity value and the match scale are going to determine how close the activations between the correct and incorrect chunks are. How large that needs to be to create the effect we want is going to depend on other settings in the model. Thus, there is not really a good guideline for determining where it should be initially, but from experience we know that it is often easier to adjust the parameters later if we start with values that allow us to see the effect each has on the results. Thus we want to make sure that we pick a value here which ensures that the similarity will make a difference in the activation values. Since the default base-level activation of chunks is 0.0 when the learning is off we are going to choose a large initial P value, like 5, to make sure that the activations will differ noticeably.

With just these settings however, the model will still not make any errors because the correct chunk will always have the highest activation and thus be retrieved. To actually get some errors we will need to also add some noise to the activation values.

5.7.7 Activation noise

In the previous unit, we saw how the activation noise affected the probability that a chunk would be above the retrieval threshold, and in this unit that is still true. In addition, since now there are multiple chunks which could all be above the threshold it is also going to affect the frequency of retrieving the correct chunk among the incorrect alternatives. The more noise there is the less likely it is that the correct chunk will have the highest activation.

As with the :mp value, choosing the initial value for the noise is not obvious because its effect is determined by other settings in the model. For this parameter however, we do have some general guidelines to work with based on past experience. For many models that have been created in the past an activation noise value in the range of 0.0-1.0 has been a good setting and for most of those the value tends to fall somewhere between 0.2 and 0.5. So, based on that, we will start this model off with a value of .5, as was used for the models of the previous unit, and then adjust things from there as needed later.

Now, given these settings, :ans .5 and :mp 5 with the similarities set as shown above, we can run the model and see what happens. Here is what we see if we just run it to collect the data:

```
> (run-siegler 100)
CORRELATION: -0.030
MEAN DEVIATION:  0.336
       0     1     2     3     4     5     6     7     8   Other
1+1  0.00  0.02  0.02  0.02  0.01  0.01  0.00  0.00  0.00  0.91
1+2  0.00  0.01  0.03  0.04  0.02  0.01  0.00  0.00  0.00  0.88
1+3  0.00  0.00  0.02  0.02  0.04  0.03  0.01  0.00  0.00  0.88
2+2  0.00  0.00  0.02  0.02  0.05  0.03  0.02  0.01  0.00  0.85
2+3  0.00  0.00  0.00  0.03  0.03  0.02  0.03  0.02  0.00  0.87
3+3  0.00  0.00  0.00  0.00  0.01  0.03  0.04  0.01  0.02  0.88
```

The model is almost never correct and most of the errors are of the type other which means that it probably did not respond. The important thing to do next is to understand why that is happening. One should not just start adjusting the parameters to try to improve the fit without understanding why the model is performing in that way.

5.7.8 Retrieval threshold and base-levels

Running the model on a few single trials and stepping through its operations shows that the problem is happening because the model is failing to retrieve chunks during all the retrieval requests, including the initial encoding of the numbers. We want the model to sometimes fail on the retrieval of the addition fact chunks, but we do not want it to be failing during the encoding phase.

So, there are two steps which we will take at this point. The first is to adjust the retrieval threshold so that we eliminate most, if not all, of the retrieval failures. This will allow us to work on setting the other parameters to match the data with the model answering the questions. Then we can come back to the retrieval threshold later and increase it to introduce more of the non-answer responses into the model. Thus, for now we will set the retrieval threshold to a value of -10.0 to make it very unlikely that any chunk will have an activation below that value.

The other thing we will do at this time is consider how to keep the number chunks from failing once we bring the retrieval threshold back up to a reasonable value. The easiest way to handle that is to increase the base-level activation of the number chunks so that the noise will be unlikely to ever take them below the retrieval threshold. The justification for doing so in the model is that it is assumed the children have a strong knowledge of the numbers and do not confuse or forget them and thus we need to provide the model with a comparable ability.

To do that we will use the set-base-levels command which works similar to the set-all-base-levels command that was used in the last unit. The difference is that for set-base-levels we can specify specific chunks instead of applying the change to all of them. Again, this seems like it is a lot of free parameters, but since we are not measuring the response time in this model all that matters is that the chunks have a value large enough to not fail to be retrieved – differences among them will not affect the error rate results as long as they are all being retrieved. We will start by assigning them a value of 10 which is significantly larger than the retrieval threshold we have now of -10 which should result in no failures for retrieving number chunks. When we increase the retrieval threshold later we may need to adjust this value, but for now we will add these settings to the model:

```
(set-base-levels
  (zero 10) (one 10) (two 10) (three 10) (four 10) (five 10)
  (six 10) (seven 10)(eight 10) (nine 10))
```

Unlike the similarities where we only needed to set the values for the numbers from 0-5 based on the task, here we need to set all of the numbers from 0-9 since any of those values is a potential sum of an addition fact in the model's declarative memory which may need to be retrieved.

After making those additions to the model running it produces this output:

```
> (run-siegler 100)
CORRELATION:  0.708
MEAN DEVIATION:  0.152
       0     1     2     3     4     5     6     7     8   Other
1+1  0.03  0.19  0.34  0.20  0.12  0.07  0.02  0.02  0.00  0.00
1+2  0.01  0.07  0.16  0.29  0.22  0.16  0.06  0.02  0.01  0.00
1+3  0.01  0.02  0.06  0.21  0.32  0.17  0.10  0.05  0.03  0.01
2+2  0.00  0.03  0.10  0.19  0.27  0.23  0.11  0.04  0.01  0.00
2+3  0.00  0.01  0.05  0.09  0.22  0.31  0.21  0.10  0.01  0.01
3+3  0.00  0.00  0.02  0.06  0.10  0.23  0.28  0.18  0.10  0.04
```

That shows a better fit to the data than the last one, though still not as good as we want, or in fact as good as it was when perfect. Looking at the trace of a few individual runs seems to indicate that the model is working as we would expect – the errors are only due to retrieving the wrong addition fact because of partial matching.

5.7.9 Adjusting the parameters

The next step to take depends on what the objectives of the modeling task are – what are you trying to accomplish with the model and what do you consider as a sufficient fit to the data. If that fit to the data is good enough, then as a next step you would then want to start bringing the retrieval threshold up to introduce some of the other responses (failure to respond) and hopefully improve things a little more. In this case however we are not going to consider that sufficient and will first investigate other settings for the :ans and :mp parameters before moving on to adjusting the retrieval threshold.

To do that we are going to search across those parameters for values which improve the model's fit to the data. When searching for parameters in a model there are a lot of approaches which can be taken. In this case, we are going to keep it simple and try manually adjusting the parameters and running the model to see if we can find some better values. When the number of parameters to search is small, the model runs fairly quickly, and one is not looking to precisely model every point this method can work reasonably well. For other tasks, which require longer runs or which have many more parameters to adjust other means may be required. That can involve writing some Lisp code to adjust the parameters and perform a more thorough search or going as far as creating an abstraction of the model based on the underlying equations and using a tool like MATLAB or Mathematica to solve those for the best values.

The approach that we use when searching by hand is to search on only one parameter at a time. Pick one parameter and then adjust that to get a better fit. Then, fix that value and pick another parameter to adjust. Do that until each of the parameters has been adjusted. Often, one pass through each of the parameters will result in a much better fit to the data, but sometimes it may require multiple passes to arrive at the performance level you desire (assuming of course that the model is capable of producing such a fit through manipulating the parameters).

Sometimes it is also helpful to work with only a subset of the parameters if you have an idea of the effects which they will have on the data. For example, in this task we know that the retrieval threshold will primarily determine the frequency with which the model gives up. Thus we are going to hold back on trying to fit that parameter until we have adjusted the others to better fit the majority of the data for the trials where it produces an answer.

Since we are starting with a noise value that was based on other tasks and our match scale value was chosen somewhat arbitrarily we will start searching across the match scale parameter. Keeping the noise value at .5 we found that a value of 16 for the match scale parameter seems to be our best fit:

```
CORRELATION:  0.942
MEAN DEVIATION:  0.074
        0     1     2     3     4     5     6     7     8   Other
1+1  0.01  0.13  0.75  0.10  0.01  0.00  0.00  0.00  0.00  0.00
1+2  0.00  0.00  0.12  0.74  0.13  0.01  0.00  0.00  0.00  0.00
1+3  0.00  0.00  0.01  0.13  0.75  0.10  0.01  0.00  0.00  0.00
2+2  0.00  0.00  0.01  0.11  0.75  0.12  0.01  0.00  0.00  0.00
2+3  0.00  0.00  0.00  0.00  0.14  0.73  0.13  0.00  0.00  0.00
3+3  0.00  0.00  0.00  0.00  0.01  0.13  0.76  0.10  0.01  0.00
```

Then, fixing the match scale parameter at 16 and adjusting the noise value we do not seem to find a value which does any better than the starting value of .5. So, we will adjust the retrieval threshold to introduce more of the other responses and hopefully improve the fit some more. Searching there finds that a value of .7 improves the fit to this:

```
CORRELATION:  0.949
MEAN DEVIATION:  0.065
        0     1     2     3     4     5     6     7     8   Other
1+1  0.00  0.07  0.74  0.09  0.01  0.00  0.00  0.00  0.00  0.09
1+2  0.00  0.00  0.10  0.70  0.08  0.00  0.00  0.00  0.00  0.11
```

```
1+3  0.00  0.00  0.00  0.10  0.71  0.09  0.01  0.00  0.00  0.08
2+2  0.00  0.00  0.01  0.12  0.66  0.12  0.00  0.00  0.00  0.08
2+3  0.00  0.00  0.00  0.00  0.13  0.68  0.09  0.00  0.00  0.10
3+3  0.00  0.00  0.00  0.00  0.01  0.08  0.70  0.08  0.00  0.13
```

One important thing to do at this point is to make sure that the model is still doing the task as we expect – that changing the parameters has not introduced some problems, like failing to retrieve the number chunks. For the current model, looking at a couple of single trial runs in detail shows that things are still working as expected. So, at this point we could go back and perform another pass through all the parameters trying to find a better fit, but instead we are going to stop and look at where our model seems to be deviating from the experimental data before trying to just find better parameters.

5.7.10 Adjusting the model

It seems that one trend in the data which we are missing is that the children seem to respond correctly more often to the smaller problems and that when they respond incorrectly the answers are more often smaller than the correct answer. There seems to be a bias for the smaller answers. This agrees with other research which finds that addition facts with smaller addends are encountered more frequently in the world.

Accounting for that component of the data is going to require making some adjustment to the model other than just modifying the parameters which we have. The research which finds that the smaller problems occur more frequently suggests a possible approach to take. The base-level activation of a chunk represents its history of use, and thus by increasing the base-level activation of the smaller plus-fact chunks we can simulate that increase in frequency and increase the probability that the model will retrieve them. This should help improve the data fit in a plausible manner.

Like the similarities, this is another instance where it looks like there are a lot of free parameters that could be used to fit the data, but again a principled approach is advised. In this case we are going to increase the base-level activation of all the small plus facts (which we have chosen to be those with a sum less than or equal to four), and we are going to give all of those chunks the same increase to their base-level activation. The default base-level activation for the plus-facts is 0. So we are going to set those chunks to have a value above that by using the set-base-levels command as we have done with the number chunks with something like this:

```
(set-base-levels
 (f00 .1)(f01 .1)(f02 .1)(f03 .1)(f04 .1)
 (f10 .1)(f11 .1)(f12 .1)(f13 .1)
 (f20 .1)(f21 .1)(f22 .1)
 (f30 .1)(f31 .1)
 (f40 .1))
```

Using the values for the other parameters found previously we will search for a base-level value which improves the data fit and what we find is that a value of .5 seems to improve things to this:

```
CORRELATION:  0.962
```

```
MEAN DEVIATION:  0.059
       0     1     2     3     4     5     6     7     8   Other
1+1  0.00  0.10  0.77  0.10  0.00  0.00  0.00  0.00  0.00  0.02
1+2  0.00  0.00  0.12  0.75  0.11  0.00  0.00  0.00  0.00  0.01
1+3  0.00  0.00  0.00  0.11  0.82  0.04  0.00  0.00  0.00  0.03
2+2  0.00  0.00  0.00  0.14  0.77  0.05  0.00  0.00  0.00  0.03
2+3  0.00  0.00  0.00  0.02  0.17  0.67  0.09  0.00  0.00  0.05
3+3  0.00  0.00  0.00  0.00  0.01  0.11  0.64  0.13  0.00  0.11
```

Given that, we will make one more pass over all the parameters (noise, match scale, retrieval threshold, and small plus-fact base-level offset) to find the final set of parameter values which are set in the given model and produce this fit to the data:

```
CORRELATION:  0.966
MEAN DEVIATION:  0.054
       0     1     2     3     4     5     6     7     8   Other
1+1  0.01  0.12  0.74  0.11  0.01  0.00  0.00  0.00  0.00  0.02
1+2  0.00  0.00  0.14  0.70  0.12  0.00  0.00  0.00  0.00  0.03
1+3  0.00  0.00  0.00  0.15  0.76  0.04  0.00  0.00  0.00  0.04
2+2  0.00  0.00  0.01  0.13  0.79  0.03  0.00  0.00  0.00  0.04
2+3  0.00  0.00  0.00  0.02  0.25  0.51  0.07  0.01  0.00  0.14
3+3  0.00  0.00  0.00  0.00  0.01  0.09  0.64  0.07  0.00  0.18
```

We could continue to search over the parameters or attempt other changes, like modifying the similarities used to something other than linear, but these results are sufficient for this demonstration. You are free to explore other changes to the parameters or the model if you are interested.

5.8 Learning from experience

The task for this assignment will be to create a model which can learn to perform a task better based on the experience it gains while doing the task. One way to do that is using declarative memory to retrieve a past experience which can be used to decide on an action to take. The complication however is that in many situations one may not have experienced exactly the same situation in the past. Thus, one will need to retrieve a similar experience to guide the current action, and the partial matching mechanism provides a model with a way to do that.

Instead of writing a model to fit data from an experiment in this unit we will be writing a model which can perform a more general task. Specifically, the model must learn to play a game better. The model will be assumed to know the rules of the game, but will not have any initial experience with the game thus must learn the best actions to take as it plays. In the following sections we will introduce the rules of the game, how the model interacts with the game, a description of the starting model, what is expected of your model, and how to use the provided code to run the game.

5.8.1 1-hit Blackjack

The game we will be playing is a simplified version of the casino game Blackjack or Twenty-one. In our variant there are only two players and they each have only one decision to make.

The game is played with 2 decks of cards, one for each player, consisting of cards numbered 1-10. The number of cards in the decks and the distribution of the cards in the decks are not known to the players in advance. A game will consist of several hands. On each hand, the objective of the game is to collect cards whose sum is less than or equal to 21 and greater than the sum of the opponent's cards. When summing the values of the cards a 1 card may be counted as 11 if that sum is not greater than 21, otherwise it must be considered as only 1. At the start of a hand each player is dealt two cards. One of the cards is face up and the other is face down. A player can see both of his cards' values but only the value of the face up card of the opponent. Each player then decides if he would like one additional card or not. Choosing to take an additional card is referred to as a "hit" and choosing to not take a card is referred to as a "stay". This choice is made without knowing your opponent's choice – each player makes the choice in secret. An additional constraint is that the players must act quickly. The choice must be made within a preset time limit to prevent excessive calculation or contemplation of the actions and to keep the game moving. If a player hits then he is given one additional card from his deck and his final score is the sum of the three cards (with a 1 counted as 11 if that does not exceed 21). If a player stays then his score is the sum of the two starting cards (with a 1 counted as 11). Once any extra cards have been given both players show all of the cards in their hands and the outcome is determined. If a player's total is greater than 21 then he has lost. That is referred to as "busting". It is possible for both players to bust in the same hand. If only one player busts then the player who did not bust wins the hand. If neither player busts then the player with the greater total wins the hand. If the players have the same total then that is considered a loss for both players. Thus to win a hand a player must have a total less than or equal to 21 and either have a greater total than the opponent's total or have the opponent bust. After a hand is over the cards are returned to the deck, it is reshuffled and another hand begins. The objective of the game is to win as many hands as possible.

There are many unknown factors in this game making it difficult to know what the optimal strategy is at the start. However, over the course of many hands one should be able to improve their winnings as they acquire more information about the current game. One complication is that the opponent may also be adapting as the game goes on. To simplify things for this assignment we will assume that the model's opponent always plays a fixed strategy, but that that strategy is not known to the model in advance. Thus, the model will start out without knowing the specifics of the game it is playing, but should still be able to learn and improve over time.

5.8.2 General modeling task description

To keep the focus of this modeling task on the learning aspect we have abstracted away from a real interface to the game in much the same way as the **fan** and **grouped** model abstracted away from a simulation of the complete experimental task. Thus the model will not have to use either the visual or aural module for acquiring the game state. Similarly, the model will also not have to compute the scores or determine the specific outcomes of each hand. The model will be provided with all of the available game state information in a chunk in the **goal** buffer at two points in the hand and will only need to make one of two key presses to signal its action.

At the start of the hand the model will be told its two starting cards, the sum of those cards, and the value of the opponent's face up card. The model then must decide whether to hit or stay. The choice is made by pressing either the H key to hit or the S key to stay. The model has exactly 10 seconds in which to make this choice and if it does not press either key within that time it is considered as staying for the hand. After 10 seconds have passed, the model's **goal** chunk will be modified to reflect the actions of both players and the outcome of the game. The model will then have all of its own card values, all of the opponent's card's values, the final totals for its hand and the opponent's hand, as well as the outcome for each player. The model must then use that information to determine what, if anything, it can or should learn from the trial before the next hand begins. The time between the feedback and the next hand will also be 10 seconds.

5.8.3 Goal chunk specifics

Here is the chunk-type definition which specifies the slots used for the chunk that will be placed into the **goal** buffer:

```
(chunk-type game-state mc1 mc2 mc3 mstart mtot mresult
                       oc1 oc2 oc3 ostart otot oresult state)
```

The slots of the chunk in the **goal** buffer will be set by the game playing code for the model as follows:

- At the start of a new hand
    - **state** slot will be the value **start**
    - **mc1** slot will hold the value of the model's first card
        - a number from 1-10
    - **mc2** slot will hold the value of the model's second card
        - a number from 1-10
    - **mstart** slot will hold the score of the model's first two cards
        - a number from 4-21 because a 1 will be counted as an 11 if there is one
    - **oc1** slot will hold the value of the opponent's face up card
        - a number from 1-10
    - **ostart** slot will hold the opponent's starting score
        - a number from 2-11 because a 1 will count as 11
    - none of the other slots will be set in the chunk

- After the players have made their decisions for the hand
    - **state** slot will be set to the value **results**
    - **mc1**, **mc2**, **mstart**, **oc1**, and **ostart** slots will be set to the same values as at the start of the hand described above
    - **mc3**

        - if the model hits the slot will be the value of the model's third card
            - a number from 1-10
        - if the model stays the **mc3** slot will not be set
    - **mtot** slot will hold the final total for the model's two or three card hand
        - a number from 4-30
    - **mresult** slot will be the model's result for the hand
        - one of **win**, **lose**, or **bust**
    - **oc2** will be the opponent's second card
        - a number from 1-10
    - **oc3**
        - if the opponent hits the slot will be the value of the opponent's third card
            - a number from 1-10
        - if the opponent stays the **oc3** slot will not be set
    - **otot** slot will be the final total for the opponent's two or three card hand
        - a number from 4-30
    - **oresult** slot will hold the opponent's result for the hand
        - one of **win**, **lose**, or **bust**

For testing, the model will be played through a series of 100 hands and its percentage of winning in each group of 5 hands will be computed. For a fixed opponent's strategy and particular distribution of cards in the decks there is an optimal strategy and it may be possible to create rules which play a "perfect" game under those circumstances. However, since the model will not know that information in advance it will have to learn to play better, and the objective is to have a model which can improve its performance over time for a variety of different opponents and different possible decks of cards. Of course, since the cards received are random for any given sequence of 100 hands the model's performance will vary and even a perfect strategy could lose all of them. Thus to determine the effectiveness of the model it will play several games of 100 hands and the results will be averaged to determine how well it is learning.

5.8.4 Starting model

The code for playing the game and a starting model for this task are in the **1hit-blackjack** file with the unit 5 materials. The given model uses a very simple approach to learn to play the game. It attempts to retrieve a chunk which contains an action to perform that is similar to the current hand from those which it created based on the feedback on previous hands. If it can retrieve such a chunk it performs the action that it contains, and if not it chooses to stay. Then, based on the feedback from the hand the model may create a new chunk which holds the learned information for this hand to use on future hands. As described below however, the feedback used by this model is not very helpful in producing a useful chunk for learning about the game – it learns a strategy of always hitting. Here are the productions from the starting model:

```
(p start
   =goal>
     isa game-state
     state start
     MC1 =c
  ==>
   =goal>
     state retrieving
   +retrieval>
     isa learned-info
     MC1 =c
   - action nil)


(p cant-remember-game
   =goal>
     isa game-state
     state retrieving
   ?retrieval>
     buffer   failure
   ?manual>
     state free
  ==>
   =goal>
     state nil
   +manual>
     cmd press-key
     key "s")

(p remember-game
   =goal>
     isa game-state
     state retrieving
   =retrieval>
     isa learned-info
     action =act
   ?manual>
     state free
  ==>
   =goal>
     state nil
   +manual>
     cmd press-key
     key =act
   =retrieval>
     mc1 nil
```

```
      action nil
    -retrieval>)

 (p results-should-hit
    =goal>
      isa game-state
      state results
      mresult =outcome
      MC1 =c
    ?imaginal>
      state free
   ==>
    !output! (I =outcome)
    =goal>
      state nil
    +imaginal>
      MC1 =c
      action "h")

 (p results-should-stay
    =goal>
      isa game-state
      state results
      mresult =outcome
      MC1 =c
    ?imaginal>
      state free
   ==>
    !output! (I =outcome)
    =goal>
      state nil
    +imaginal>
      MC1 =c
      action "s")

 (p clear-new-imaginal-chunk
    ?imaginal>
      state free
      buffer full
    ==>
    -imaginal>)
```

The operation of most of those productions should be fairly straight forward, but there is something different in the actions of the **remember-game** production which is worth noting. On its RHS we see these actions:

```
   =retrieval>
```

```
        mc1 nil
        action nil
    -retrieval>
```

Before clearing the chunk from the **retrieval** buffer all of the slots are explicitly removed (setting a slot to **nil** removes it from the chunk). That is done to prevent it from merging back into declarative memory and strengthening the chunk which was retrieved. The reason for that is because the chunk which was retrieved may not be the best action to take in the current situation either because it was retrieved due to noise or because the model does not yet have enough experience to accurately determine the best move. So, the model erases that information and waits for the feedback on the hand before creating a new chunk to represent the action to take on this hand. If it did not do that, then it could continue to strengthen and retrieve a "bad" chunk just because it was created and retrieved often early on in its learning. There are other ways in which that can be managed, but this simple mechanism is sufficient for the task being modeled here.

Because there are many more potential starting configurations than hands which will be played this model uses the partial matching mechanism to allow it to retrieve a similar chunk when a chunk which matches specifically is not available. The given code provides the model with similarity values between numbers by using a Lisp function. This is done through the use of a "hook function" parameter in the model. A hook function parameter allows the modeler to override or modify an internal computation through Lisp code and there are several which can be specified for a model. In this case we are setting the :sim-hook parameter to compute the similarity values for the model. This is being done because the set-similarities command only allows the modeler to set the similarity value between chunks, but here we are using numbers to represent the card values and hand totals. Even if we had used chunks to represent the card values however it would still have been easier to use the hook function to compute the similarity values instead of having to explicitly specify all of the possible values for the similarities between the numbers which can occur while playing the game – essentially all the possible pairs for numbers from 1 to 30.

The equation that is used to set the similarities between the card values is:

$$Similarity(a,b) = -\frac{abs(a-b)}{\max(a,b)}$$

This ratio has two features which should work well for this task and it corresponds to results found in the psychology literature. First, the similarity is relative to the difference between the numbers so that the closer the numbers are to each other the more similar they are. Thus, 1 and 2 are more similar than 1 and 3. The other feature is that larger numbers are more similar than smaller numbers for a given difference. Therefore 21 and 22 are more similar than 1 and 2 are.

There are several other parameters which are also set in the starting model. Those are divided into two sets. The first set is those which control how the model is configured (which learning mechanisms are enabled and how the system operates), and the second set is those which control the parameters of the mechanisms used in the model. This is the first set of parameters:

```
(sgp :esc t :bll .5 :ol t :sim-hook number-sims :er t :lf 0)
```

It enables the subsymbolic components of ACT-R. It turns on the base-level learning for declarative memory with a decay of .5 (the recommended value) and specifies that the optimized learning equation be used for the base-level calculation. It specifies the function which will compute the similarity values. Randomness is enabled to break ties for activations and during conflict resolution. Finally, the latency factor is set to 0 so that all retrievals complete immediately which is a simplification to avoid having to tune the model's chunk activation values to achieve appropriate timing since we aren't matching latency data. Those settings should also be used in the assignment model which you write.

The other set of parameters in the starting model specifies the things which you may want to modify:

```
(sgp :v nil :ans .2 :mp 10.0 :rt -60)
```

In addition to the :v flag to control the trace it sets the activation noise to a reasonable value. The mismatch penalty for partial matching is set at 10 and the retrieval threshold is set very low so that the model should always be able to retrieve some relevant chunk if there are any. These values worked well for the solution model, but may need to be adjusted for your model.

5.8.5 The Assignment

The assignment for the task is to create a model which can learn to play better in a 100 hand game without knowing the details of the opponent or the distribution of cards in the deck in advance. Thus, it must learn based on the information it acquires as it plays the game.

Although the specific information learned by the starting model does not do a good job of learning to play better it does represent a reasonable approach for a model of this game. The recommended way to approach this assignment is to modify that starting model so that it learns to play better. You are not required to use that model, but your solution must use partial matching and it must be able to learn verses a variety of opponents and with different distributions of cards in the deck – it should not incorporate any information specific to the strategy of the default opponent provided or the default distribution of cards in the deck.

If you choose to use the starting model, then the thing that you will need to change about it to make it better learn to play the game is how it interprets the feedback so that it creates chunks which have appropriate information about the actions it should take based on the information that is available.

Specifically, you will need to do the following things:

- Change the learned-info chunk-type to specify the slots which hold the information your model needs.

- Change the start production to retrieve a chunk given the information you have determined is appropriate.
- Change the results-should-hit and results-should-stay productions to better test the information available at the end of the game to decide on a good action to learn. This may require adding additional productions as well, or setting the productions' utilities to favor some actions over others.

With an appropriate choice of initial information and feedback that should be sufficient to produce a model which can learn against a variety of opponents. There are other things which you could do that may improve that model's learning, and some of those are listed below. It is strongly recommended that you get a simple model which can learn to play the game using the basic operations described above before attempting to improve it with any of these or other mechanisms:

- Change the noise level and the mismatch penalty parameters to attempt to adjust the learning rate or flexibility of the model.
- Add more productions to analyze the starting position either before or after retrieving an appropriate chunk.
- Provide a strategy other than always staying when no relevant information can be retrieved.

5.8.6 Running the Game and Model

There are three functions that can be used to run a model through the game against an opponent who is controlled by Lisp code: play-hands, run-blocks, and show-learning. Each is described along with examples below.

play-hands

Play-hands takes one required parameter which is the number of hands to play and an optional parameter which controls whether it prints out the details of each of those hands. It returns a list of four items. The items in the list are the counts of the model's wins, the opponent's wins, the times when both players bust, and the times when they are tied.

Here is an example of running it without the optional parameter:

```
> (play-hands 4)
(1 3 0 0)
```

Here is an example of running it with the optional parameter to show the details of the hands played:

```
> (play-hands 5 t)
```

```
Model:  9  6  -> 15 (LOSE)   Opponent:  6  8  4 -> 18 (WIN )
Model:  7  1  6  -> 14 (LOSE)   Opponent:  7 10 -> 17 (WIN )
Model:  4 10  7  -> 21 (WIN )   Opponent: 10  2  4 -> 16 (LOSE)
Model:  8 10 10  -> 28 (BUST)   Opponent:  8  4 10 -> 22 (BUST)
Model:  1 10 10  -> 21 (WIN )   Opponent: 10  4  9 -> 23 (BUST)
(2 2 1 0)
```

An important thing to note is that play-hands does not reset the model. Thus it will retain any information which it has learned from one call to the next, and if you want to start the model over you will have to reset it explicitly.

run-blocks

Run-blocks takes two parameters which are the number of blocks to run and the number of hands to run in each block. It runs the model through those hands and returns the list of results per block where each block is represented by a list as returned by play-hands. Here is an example with running two blocks of 5 hands each:

```
> (run-blocks 2 5)
((1 3 1 0) (2 2 1 0))
```

Note that like play-hands the run-blocks function also does not reset the model.

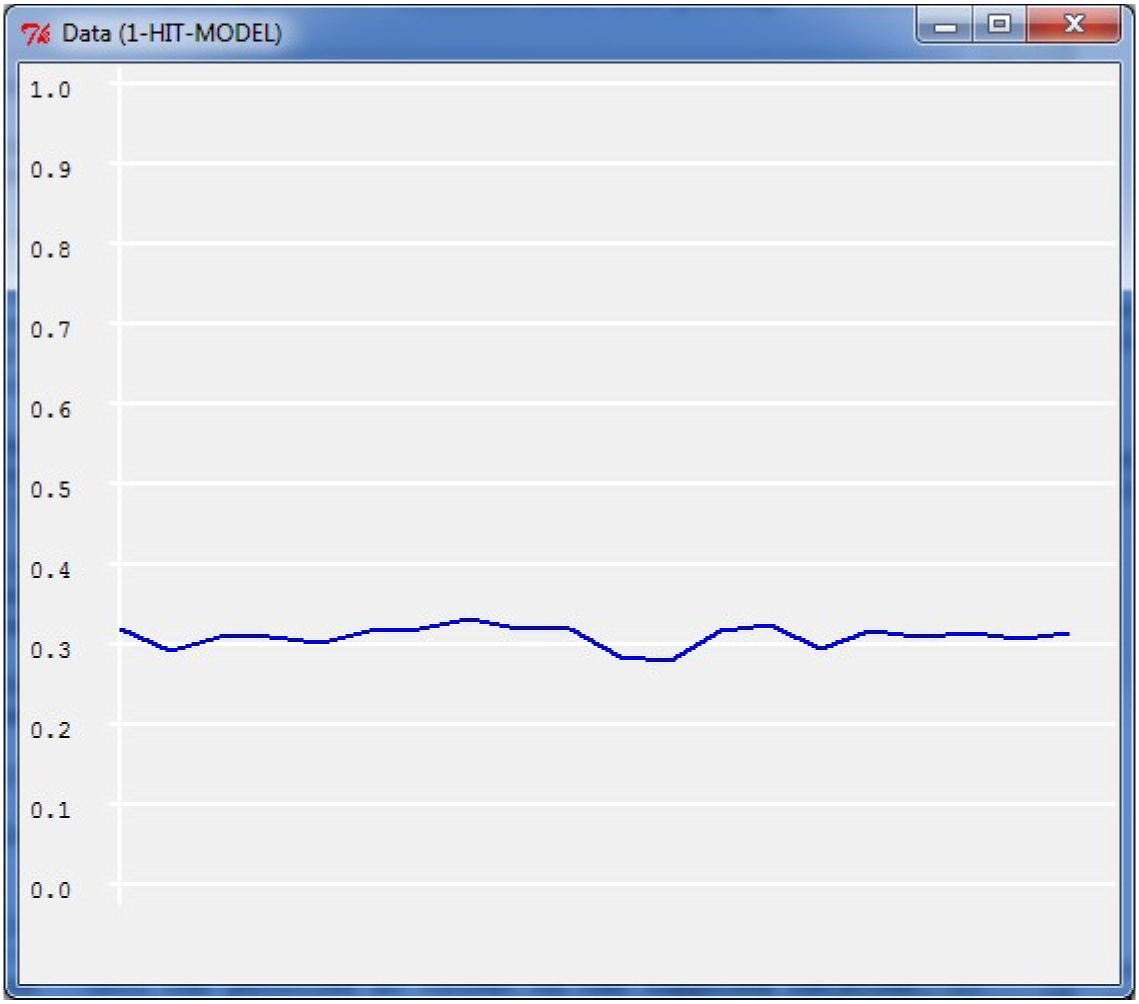
show-learning

Show-learning is used to run a model through multiple 100 hand games for analysis. One game of 100 hands is equivalent to using run-blocks for 20 blocks of 5 hands. It takes one required parameter which is the number of games to run the model through and then average over. It also takes an optional parameter to indicate whether or not a graph of the results should be drawn and an optional parameter to indicate a function for specifying the game details.

This resets the model before running each 100 hand game. The results of those games are collected and averaged as 4 blocks of 25 hands and as 20 blocks of 5 hands. It returns a list of two lists. The first list is the percentage of wins in the 4 blocks of 25. This list should give a quick indication of whether or not the model is improving over the course of the game. The second list is the percentage of wins considering 20 blocks of 5 hands to provide a more detailed description of the learning. If the first optional parameter is not given or is specified as t, then an experiment window will be opened and the detailed percentage data will be displayed in a graph. Here is an example call and resulting graph of a run of the example model:

```
> (show-learning 200)

 ((0.3056 0.32039997 0.30020002 0.31199998) (0.318 0.291 0.308 0.31 0.301 0.316 0.319
0.331 0.318 0.318 0.285 0.28 0.316 0.325 0.295 0.317 0.308 0.314 0.306 0.315))
```



If you do not want to see the graph then specifying the second parameter as nil will suppress it:

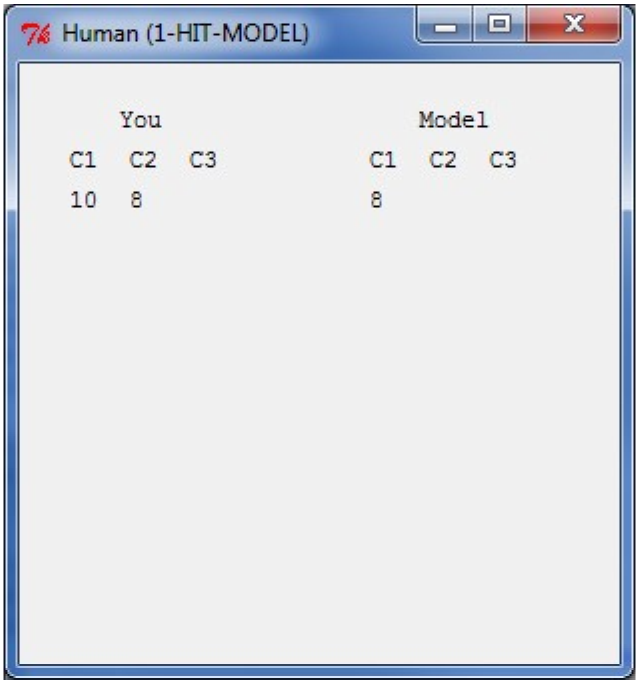
```
> (show-learning 200 nil)

 ((0.3096 0.3012 0.30440003 0.296) (0.308 0.31 0.306 0.326 0.298 0.29 0.302 0.298
0.288 0.328 0.318 0.316 0.28 0.314 0.294 0.294 0.286 0.312 0.286 0.302))
```

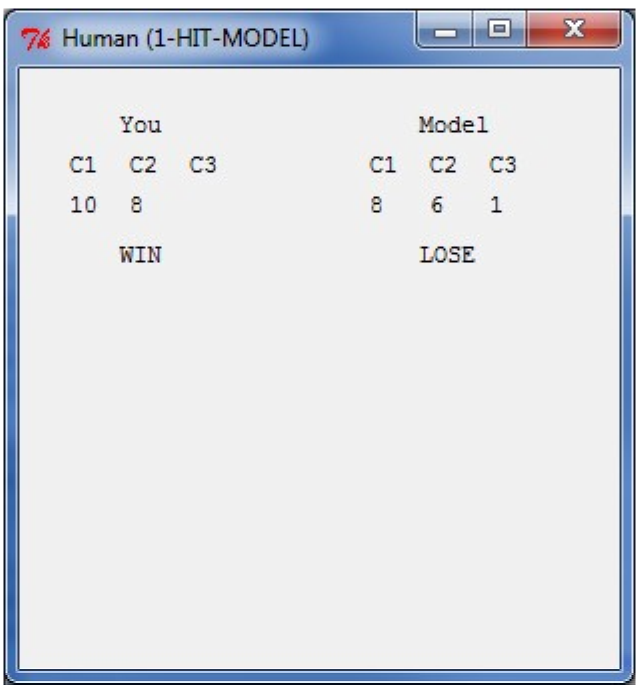
How to use the other optional parameter is described in this unit's experiment description text.

There is one other function which can be used to run the model called play-against-model. This function is similar to play-hands except that instead of running the Lisp code for an opponent it opens a window like that shown below which allows you to play against the model. It requires a number of hands to play as a parameter along with an optional parameter indicating whether or not to print the hand results in the Listener just like play-hands.

At the start you will see your starting cards and the model's face up card for 10 seconds in which time you must respond by pressing h to hit or s to stay.



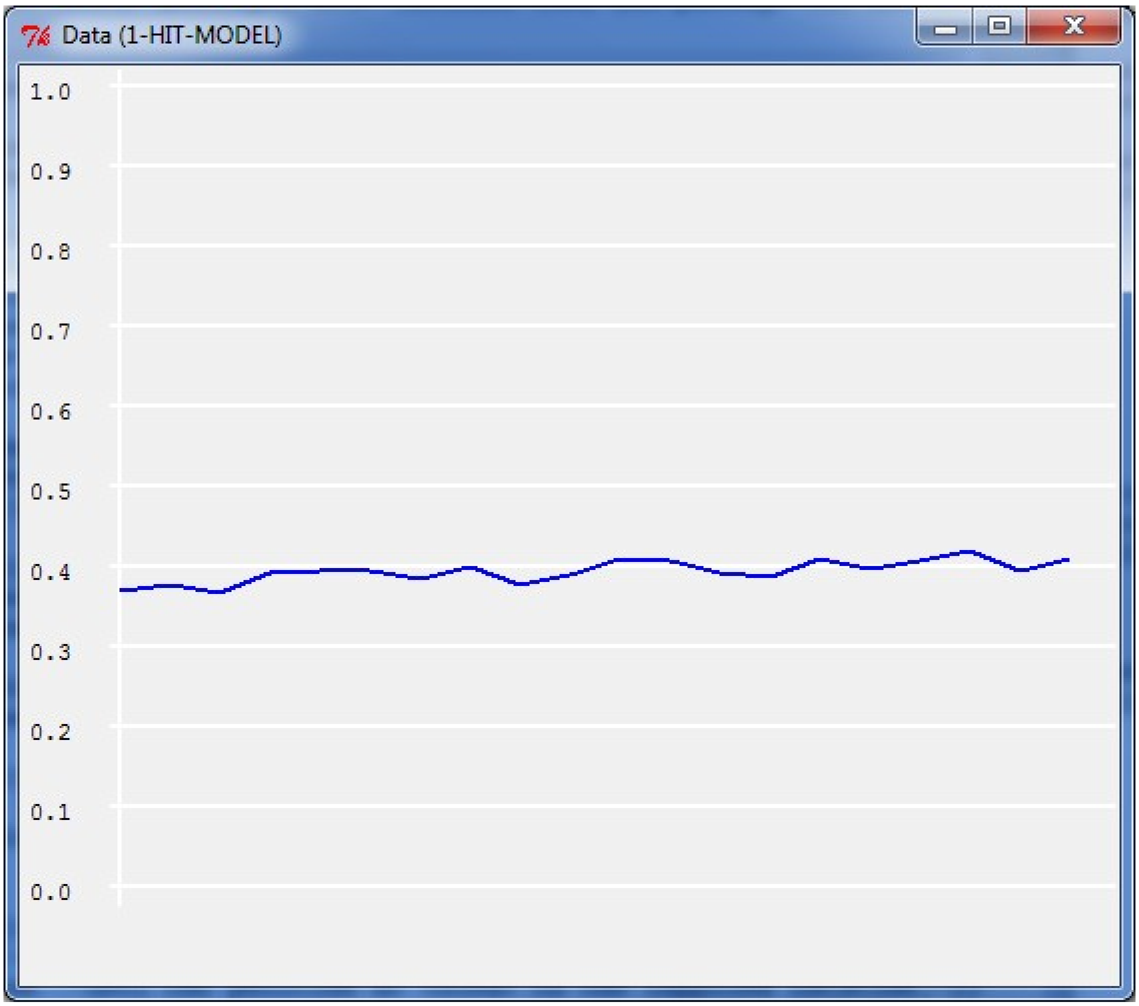
After 10 seconds pass it will then show all of your cards and all of the model's cards along with the results for 10 seconds before going on to the next hand.



5.8.7 The Default Game

The default game your model will be playing is an opponent who always stays with a score of 15 or more and both decks have an effectively infinite number of cards in a distribution like a normal deck of playing cards (an equal distribution of cards with the values from 1-9 and four times as many cards with a value of 10). Under those circumstances the optimal strategy against that opponent would win about 46% of the time and choosing randomly wins about 32% of the time.

The reference solution model is able to improve from winning about 37% of the time in the first block to winning around 40% in the final blocks on average over the 100 hands as shown in this graph from running 500 games.



It is also able to learn against other opponents and when the distribution of cards in the deck differs. Your model should show similar or better performance for that default game and also still be able to learn in other situations i.e. just encoding an optimal strategy for the default opponent and deck distributions into your model is not an adequate solution to the task.

After producing a model which learns to play the default game you may want to try testing it with different opponents or other decks. The experiment code which runs the task has some flexibility built into it that allows for modifying the game that is played. Details on how to change the game code and some suggestions for other game situations are found in the experiment description text for this unit.

References

Anderson, J. R. (1974). Retrieval of propositional information from long-term memory. Cognitive Psychology, 5, 451 – 474.

Siegler, R. S., & Shrager, J. (1984). Strategy choices in addition and subtraction: How do children know what to do? In C. Sophian (Ed.), Origins of cognitive skills (pp. 229-293). Hillsdale, NJ: Erlbaum.