Lecture 4: Declarative Learning

Thinking about Memory

- Suppose you are a librarian who has to move your library to a smaller building — and throw away 50% of the books
- How do you choose which books to keep, and which books to throw away?

This Week

- We look at one of the most important components of human cognition — Memory!
- Thanks to Jelmer Borst and Niels Taatgen for the basis for these slides!
Thinking about Memory

- New York Times word usage

- NY Times, parental speech, email
  - Probability of seeing word on 101th episode as a function of # episodes since last occurrence

- Probability on a log-log scale

- Ebbinghaus (1850-1909)
  - Forgetting
  - Practice
**Thinking about Memory**

- **Ebbinghaus (1850-1909)**

**Forgetting**

\[ P = 47.56 	imes T^{-1.26} \]

**Practice**

\[ P = 59.1 	imes S^{-1.44} \]

---

**Declarative Memory**

- Declarative memory = memory of facts
  - \(3+2=5\)
  - The past tense of “go” is “went”
  - Last week I had blueberry soup (episodic)

- In ACT-R, these are all chunks

---

**Activation**

- Activation is a “subsymbolic” feature of a chunk that determines how easily it’s recalled
- Activation determines...
  - Whether a chunk can be retrieved
  - Which chunk is retrieved
  - How long it takes for a chunk to be retrieved

\[ A_i = B_i + C_i + M_j + \varepsilon \]

- base-level activation
- spreading activation
- mismatch penalty
- noise
Base-Level Activation

- Reflects the odds that you need a chunk based on frequency and recency
- Remind me, what are “odds”?
  - Odds represent the probability of an event occurring compared to it not occurring
    \[ \text{Odds} = \frac{p}{1-p} \]
  - E.g., odds of 3 means that an event is 3 times as likely to occur than not to occur
  - E.g., odds of 1/1 (or 1:1 or 1-to-1) = 50% / 50%
  - (PS: Betting odds are often expressed as odds against something happening — like 12:1 odds for the Eagles winning the Super Bowl)

\[ \text{Odds}(t) = (t - t_i)^{-d} \]

For a single time the chunk is used...

\[ \text{Odds}_k(t) = \sum_{k=1}^{n} (t - t_k)^{-d} \]

- What counts as a “use”?
  - Chunk creation
    - add-dm, perceptual buffer, +goal>
    - clearing the buffer: -imaginal>
  - Retrieval and harvesting
    - when a retrieval is used in =retrieval>
  - Chunk merging
    - a new chunk merged with an older, identical chunk

\[ t_i \] time since last use
\[ d \] decay parameter
Base-Level Activation

\[ B_i(t) = \ln \left( \sum_{k=1}^{n} (t - t_k)^{-d} \right) \]

- \( n \) presentations
- decay parameter
- time since \( n \)th presentation

Optimized Learning

- For every retrieval, ACT-R has to calculate the activity of matching chunks by summing over all presentations
- This is computationally expensive
- And perhaps not completely plausible — it implies (well, maybe) that we store all uses somehow
Optimized Learning

$$B_i(t) = \ln \left( \sum_{k=1}^{n} (t - t_k)^{-d} \right)$$

$n$ presentations

Time since first presentation

decay parameter

Activation

$$A_i = B_i + \varepsilon$$

$$B_i(t) = \ln \left( \sum_{k=1}^{n} (t - t_k)^{-d} \right)$$

base-level activation

noise

Activation Noise

- We assume that activation, like most things, has some associated noise in the system
- “permanent” vs. “now”

$$\varepsilon = \varepsilon_{\text{now}} + \varepsilon_{\text{permanent}}$$

added at retrieval

associated with a chunk
Activation Noise

\[ \mathcal{E} = \mathcal{E}_{\text{now}} + \mathcal{E}_{\text{permanent}} \]

Logistic Distribution

\[ \mu = 0 \]
\[ \sigma^2 = \frac{\pi^2}{3} s^2 \]

Retrieval Time

\[ \text{Time} = Fe^{-A_i} \]

- \( F \) is a parameter giving the time scale (called the latency factor)
  - defaults to 1.0
- \( A \) is the activation of the retrieved chunk
- Retrieval Threshold \( \tau \)
  - \( A \geq \tau \), or retrieval fails (causes “state error”)
  - defaults to 0.0

Retrieval Probability

- Given a chunk’s activation, what’s the probability that it can be recalled?
  - Based on threshold and noise

\[ \text{recall probability}_i = \frac{1}{1 + e^{(\tau - A)_i}} \]
Retrieval Probability

Recall probability, \( P_r \), is given by:

\[
P_r = \frac{1}{1 + e^{-A_i}}
\]

Activation

Parameters

\[
A_i = B_i + \varepsilon \quad \text{if } > 0
\]

\[
B_i(t) = \ln \left( \sum_{k=1}^{n} \left( t - t_k \right)^{-d} \right)
\]

\[
\varepsilon = \varepsilon_{\text{new}} + \varepsilon_{\text{permanent}}
\]

Spreading Activation

- Spreading activation reflects increased probability of needing a chunk if it is associated with the current context.
- What's the “context”? The buffers.

\[
A_i = B_i + C_i + M_i + \varepsilon
\]
**Spreading Activation**

The goal context spreads activation to 3, 8 — which boosts activation and retrieval of fact: $3 + 5 = 8$

\[ C_i = \sum_k \sum_j W_{kj} S_j \]

- **Activation spread from a buffer**

**Paired Associates**

- The example task in Unit 4
- Task works as follows...
  - see alternating <word> and then <digit>
    - e.g., card → 1, face → 3, game → 4, ...
  - recall what digit is associated with the word: when you see the word, type the digit
- Let's try it...
tent

2

zinc

9
Paired Associates

- What’s your prediction for
  - Response accuracy?
  - Response time?

Paired Associates

- Experiment results

![Graphs showing accuracy and response time over trials](image)
Paired Associates

**Word**

(p read-probe =goal>
  isa goal
  state attending-probe
  isa text
  value =val
)

=>
+imaginal>
  isa pair
  probe =val
  +goal>
  state testing
}

**Number**

(p associate =goal>
  isa goal
  state attending-target
  isa text
  value =val
  +imaginal>
  isa pair

=>
+imaginal>
  answer =val
+imaginal>
  state start
+visual>
  isa clear

Paired Associates

**Read Word**

(p read-probe
  isa goal
  state attending-probe
  isa text
  value =val
)

=>
+imaginal>
  isa pair
  probe =val
+goal>
  state testing

**Retrieval Error**

(p cannot-recall
  isa goal
  state testing

=>
+retrieval>
  isa goal
  state read-study-item
  isa clear

Paired Associates

**Read Word**

(p read-probe
  isa goal
  state attending-probe
  isa text
  value =val
)

=>
+imaginal>
  isa pair
  probe =val
+retrieval>
  isa pair
  probe =val
+goal>
  state testing

**Retrieval**

(p recall
  isa goal
  state testing
  =retrieval>
  isa pair
  answer =ans

=>
+manual>
  isa free
+manual>
  isa press-key
  key =ans
+goal>
  state read-study-item
+visual>
  isa clear

Paired Associates

**Model results**

- **Accuracy**

- **Response Time**
Paired Associates

Model results

Accuracy

Response Time

$\tau = -2$

Task Interfaces

So far, the only code we’ve looked at is the ACT-R model code.
- this is the most important code
- but for any real task, we need task code

Java ACT-R provides a common task interface for all ACT-R models
- based on Java Swing = GUI components in JFC
- Java ACT-R builds on top of this to provide additional functionality
Task Interfaces

- Most importantly: You’ll need “ACT-R.jar” as an external library to your Java code
  - comes with the system download
  - in Eclipse, the project looks something like this

Starter.java

- Not required — only for convenience in loading the model you specify by default
- Must be placed in package “actr.tasks”
  - (just this file - other code can be anywhere)

```java
package actr.tasks;
import java.io.File;
import actr.env.Core;
public class Starter implements actr.env.Starter {
    public void startup(Core core)
    {
        // load the specified file at startup (replace with your own file path/name)
        core.openFrame(new File("/Users/ << file path >> /Thread1.actr"));
    }
}
```

SampleTask.java

```java
package cs680;
import actr.task.*;
public class SampleTask extends Task // extends actr.task.Task
{
    TaskLabel labels[] = new TaskLabel[5];
    public SampleTask ()
    {
        // call the parent class for initialization
        super();
        // create new text labels that will be visible to ACT-R
        for (int i = 0; i < labels.length; i++)
        {
            // create a label: name, x, y, width, height
            labels[i] = new TaskLabel(Integer.toString(i+1), 30+30*i, 50, 20, 20);
            // add it to the panel
            add(labels[i]);
        }
    }
    public void start ()
    {
        // add a sound event that will begin at time=0.0
        addAural(0.0, "ring", "ring", "ring");
        // refresh ACT-R’s visual system with all currently visible labels
        refreshVisible();
    }
}
```

Task Details

- “Task” extends Java Swing’s “JPanel”
  - has a “null” layout, so you should specify absolute <x,y> coordinates for each component
  - standard Swing components can be added
    - e.g., JButton, JTable, JLabel, etc.
  - ... but ACT-R won’t be able to “see” or interact with these components
  - for ACT-R interaction, use the provided Task components
Task Details

- TaskComponent : interface for all components with which ACT-R can interact
  - TaskLabel
    - kind = "text", value is the text itself
  - TaskButton
    - kind = "oval", value is the button’s text label
    - doClick() is called when ACT-R clicks it
      - of course, this can be overridden to specify functionality if you’d like
  - TaskCross, TaskLine, ...
  - or specify your own by implementing the TaskComponent interface

Task Details

- Basic Task methods
  - Task() : override the constructor
    - remember to call super() for Task initialization
  - start() : override to initialize your task
  - add() : add a TaskComponent to the interface
  - processDisplay() : refresh ACT-R’s vision
  - analyze() : analyze results of task simulations

- Updating methods
  - addUpdate (dt) : schedules an update in dt secs
  - addPeriodicUpdate (dt) : adds a regularly scheduled update every dt seconds
  - update (time) : called for scheduled updates

More useful methods

- addAural() : add sound event for ACT-R to hear
- addEvent() : add a generic ACT-R event
  - you specify the function by extending actr.model.Event
- typeKey() : override to handle a model keystroke
- speak() : override to handle a spoken string
- getModel() : returns the ACT-R model
  - sometimes useful to extract parameter values, etc.

Check out the JavaDocs for more info!
- located in the downloaded code directory

---

### U4Paired.java

```java
package actr.tasks.tutorial;
import java.util.*;
import actr.task.*;
public class U4Paired extends Task {
    //--- Task Code ---//
    TaskLabel label;
    double lastTime = 0;
    int pairIndex = 0,
    pairItem = 0;
    String[][] pairs = {
        {"bank","0"},
        {"card","1"},
        {"dart","2"},
        {"face","3"},
        {"game","4"},
        {"hand","5"},
        {"jack","6"},
        {"king","7"},
        {"lamb","8"},
        {"mask","9"},
        {"neck","0"},
        {"pipe","1"},
        {"quip","2"},
        {"rope","3"},
        {"sock","4"},
        {"tent","5"},
        {"vent","6"},
        {"wall","7"},
        {"xray","8"},
        {"zinc","9"}
    };
    int iteration = 0;
    final int runIterations = 8;
    String response = null;
    double responseTime = 0;
    Trial currentTrial;
    Vector<Trial> trials =
        new Vector<Trial>();
    class Trial {
        int responses = 0;
        int responsesCorrect = 0;
        double responseTotalTime = 0;
    }
}...
```
```java
public U4Paired ()
{
    super();
    label = new TaskLabel("-", 150, 150, 40, 20);
    add(label);
}

public void start ()
{
    iteration = 0;
    lastTime = -10;
    pairIndex = 0;
    pairItem = 0;
    response = null;
    responseTime = 0;
    currentTrial = new Trial();
    Utilities.shuffle(pairs);
    addPeriodicUpdate(5.0);
}

public void typeKey (char c)
{
    response = c + "";
    responseTime = getModel().getTime() - lastTime;
}

...
Alphabet Arithmetic

- **Goal:** study learning of facts from scratch
- **Verify equations with letters, e.g.**
  - \( G + 3 = J \)
  - \( N + 4 = S \)
- **Strategy #1: Counting**
  - Step from first letter, the given number of times
  - Takes more time as the addend grows
  (addends were +2, +3, +4)

---

Alphabet Arithmetic

- **What we’d expect from counting...**

---

Alphabet Arithmetic

- **The actual results...**

---

Alphabet Arithmetic

- **Strategy #2: Retrieval**
  - Remember what you’ve counted previously, and just recall it
  - A more likely strategy after practice
  - Predicts no effect of the addend size — speed is based on prior frequency
Assignment 2

- Extend the alpha-arithmetic model to include the retrieval strategy
  - Model: Start with the given counting model
  - Code: Use the given Java task code
    - You don’t need to modify it
    - But you need to compile and run it with ACT-R.jar
    - Please feel free to try new things with it — you’ll be developing task code in the next assignment