Problem Set 2: Cognitive Architectures and Rule-based Systems
CS328, Spring 2016, Anna Rafferty

You are welcome to talk to others about the problems on this problem set, but you should read and attempt each problem prior to discussing it with others. The write up must be your own work. That means that you should write your answers by yourself, without looking at notes from sessions with collaborators, and you must be able to explain any answer you write down. You should list anyone you collaborated with in your collaboration file. See the handout on homework to find out more details.

Due date: All questions are due at 11PM on Friday, 8 April 2016. Please submit a zip file on Moodle. For this assignment, you should zip up the directory containing your notebooks (CS328-PS02-01.ipynb and CS328-PS02-02.ipynb), your python modules (actr.py and productionrules.py), and the data directory that's included with the starter files. Additionally, as for all assignments, include your collaborations.txt file in this directory; read the homework policies to learn more about what this file should include.

Starter files: The starter files for this assignment are available here.

1. ACT-R. In this problem, you’ll build a simplified ACT-R learner to recognize the past-tense of English verbs. For this problem, you’ll put your answers in CS328-PS02-01.ipynb (a notebook with the correct headers is included), and a python module named actr.py (you should create this file).

Included with the problem set are three files relevant to this problem: verbChunks.txt, childesFrequency.txt, and lags.txt. verbChunks.txt contains 10 English verbs. The first column is the name of the verb, the second column is the stem of the past tense of the verb, and the third column is the suffix of the past tense. childesFrequency.txt is the parental frequency count of each past tense verb in the CHILDES corpus (Li & Shirai, 2000). CHILDES is a collection of transcripts of speech between children and parents (and occasionally experimenters); parental frequency count is how often a particular word was used by an adult. lags.txt is a collection of hypothetical time lags that might have elapsed since seeing instances of each of the past tense forms. Each of the ten columns in lags corresponds to a past tense form from chunks. The time lag in each row of lags is from a different instance of seeing that chunk. The number of lags for a chunk is (roughly) equal to the number of times that word was seen divided by 10; the lags are to a sampling of the occurrences rather than all of them, but you can generally ignore this fact when working through this problem.

(a) The baseline activation of a chunk for an ACT-R learner is:

\[
activation = \ln \sum_{lag \in lags} lag^{-d},
\]

where the sum is over the collection of time lags for a single verb, \(d\) is the decay parameter of the model, and \(\ln\) is a logarithm with base \(e\) (“natural log”). In your actr.py module, create:

- A function `baseline_activation` that takes two parameters: a float \(d\) and a list of lags, and returns the baseline activation for a chunk with the given lag times.
- A function `compute_all_baselines` that takes one filename as a parameter: this should be a file of time lags (like lags.txt, although as with all homework specifications, anything in the same format should work) with each column corresponding to one verb. This function should return a list of all baseline activations. Assume that we use a default decay parameter of 0.5; we’ll keep this decay parameter the same for the remainder of the problem.

In the notebook section “Baseline activations”, include a cell that uses your module to compute the baseline activations for the verbs in the Childes data. This cell should also create a bar graph showing the baseline activations for these verbs. Like all graphs you create for this class, you should make sure both axes are labeled and the graph should have a title.

(b) Examine the baseline activation of “hide” versus “smash”. Which one has a higher activation? In your notebook, below the cell that graphs the baseline activations, explain why this word has a higher activation, making reference both to how the activation is computed and the values for the time lags. Your explanation should help the reader make sense of the relationships between time lags and activations - it shouldn’t just say something like “the formula gives a higher number for word1 versus word2.”

(c) In the section named “Relating baseline activation and frequency”, explain the relationship between the frequency of each verb (in childesFrequency.txt) and the baseline activations. In general, what sort of pattern do you observe and why? Include in this section the code to create a scatter plot where the frequency of each verb is on the x-axis and the baseline activation for that verb is on the y-axis.
(d) The activation of a chunk controls how easy it is to retrieve from memory. We assume an activation threshold of $\tau = 0.3$: a chunk can only be retrieved from memory if its activation is greater than 0.3. Given the activations you computed in part (a), which past tense forms can be retrieved from memory by this learner? In the “Retrieval” section, include code that computes and displays your answer. Imagine there’s a word that has only been seen once (so there’s exactly one time lag). For how long after exposure could the learner retrieve this word from memory, assuming the default decay and activation thresholds? Explain your answer by showing your work (e.g., mathematical or coding work, which may also be supported by text explanations).

(e) Imagine there are two different words. Both have been seen by the learner ten times. Create two different series of time lags such that the first word is retrievable from memory but the second word is not. Write down these timelags in the section named “Time lags and retrieval”. In that section, also include a cell that demonstrates that the first word is retrievable from memory and the second is not.

(f) For chunks that have high enough activation to be retrieved, retrieval time $t$ is computed as $t = F \cdot \exp(-fA)$, where $A$ is the baseline activation and $F, f$ are parameters of the model. $\exp$ means the mathematical constant $e$ raised to a particular power; for instance, $\exp(-fA) = e^{-fA}$. In your actr.py module, write a function retrieval_time that takes three parameters: (1) the activation, (2) $F$, and (3) $f$. This function should return the retrieval time for the chunk. In the “Retrieval time” section, use $F = 0.5$ and $f = 0.25$ to compute the retrieval time for all the past tense forms that have high enough activation to be retrieved. Print out the retrieval times, labeled with which verbs they refer to, in your notebook.³

(g) Experiment with different values of $f$, restricting yourself to values in between 0 and 1. Also in the “Retrieval time” section, describe the pattern of how the magnitude of this parameter affects the retrieval time. You should include evidence from your experimentation to support the pattern you describe.

(h) An ACT-R learner only retrieves chunks from memory when retrieval is requested by a production rule. Production rules are selected according to their utility. The utility of a production rule $i$ is given by $U_i = P_i \cdot G - C_i$, where $P_i$ is the probability of success, $G$ is the value of the current goal, and $C_i$ is the cost for using this rule (measured in units of time). We will assume that the value $G$ of creating a past tense form is equal to 5. We will further make the simplifying assumption that the learner only has two possible rules for creating past tense forms: the regular past tense rule (“add -ed”) and the retrieval rule.

The regular past tense rule always succeeds in producing a past tense form, so $P_i = 1$ for this rule. The cost $C_i$ for the regular rule is stipulated as a phonetic post-processing cost of 1.2. In your module actr.py, write a function regular_utility that takes no parameters and uses these default values to compute the utility of the regular rule; the function should return the rule’s utility.

(i) To compute the retrieval rule’s utility, we need to average over the words that our learner will want to produce. We assume that our learner tries to produce forms in the lexicon according to their frequencies. This probability of success, $P_i$, can be computed as the ratio of the total frequency of words above the activation threshold $\tau$ to the total frequency of all words. You also need to know the average cost $C_i$, which is equal to the average retrieval time. The average retrieval time is the average of the retrieval times for all words in the vocabulary that can be retrieved, weighted by their frequency. For instance, assume there are two words in the vocabulary: word1 has retrieval time 2 and frequency 10, and word2 has retrieval time 1 and frequency 1. The average retrieval time is $(2 \times 10 + 1 \times 1)/(10 + 1)$. Words that cannot be retrieved do not count towards the cost.

In actr.py, write a function retrieval_utility which takes two parameters: (1) the filename for a frequency file (like childesfrequency.txt) and (2) the filename for a time lags file (like lags.txt). This function should return the utility of the retrieval rule based on the given frequencies and lags. When calculating retrieval times, assume $F = 0.5$, $f = 0.25$, and $G = 5$.

Use your function to compute the utility of the retrieval rule given the data you have been provided. Include the utility of the regular rule and the retrieval rule in the “Utilities” section.

(j) Assume the learner always starts by using the rule with higher utility, and then if the first rule fails (i.e., can’t be executed with this word), uses the rule with lower utility. Given the data distributed with the problem set, which past tense forms will your learner produce correctly? Which past tense forms will your learner over-regularize? Over-regularizing is when a rule like “add -ed” is used for verbs it shouldn’t be, such as saying “I thinked” rather than “I thought.” Write your answers in the “Utilities” section of your notebook.

(k) Some studies of children’s overregularization have found that children start off not overregularizing the verbs they know, then overregularize for a period of time, and then finally consistently produce correct verbs. Could this pattern be consistent with this simplified ACT-R model? Explain why or why not, making reference to trends in the data that you might expect to see and how this would or would not lead to the proposed pattern. Put your answer in the final section of the notebook named “Overregularization and ACT-R.”

³To reload the module in the notebook, you might restart the kernel and then rerun all cells.
2. Cognitive architectures often use production rules for reasoning. Production rules are logical implications in the form \( if \; \alpha, \; then \; \beta; \) written in logic, \( \alpha \rightarrow \beta. \) If the system believes that \( \alpha \) is true, then it can conclude that \( \beta \) is true using this rule.

We'll implement a production rule system in Python. We first need a representation of the system's rules and its beliefs. To start, we'll keep things relatively simple and assume each rule consists of exactly one statement implying another statement, where the statements contain no logical connectives. We'll represent this as a two-item tuple, with the first item equal to the antecedent (if part) and the second item equal to the consequent (then part). For instance, the logical rule “If gas meter at zero, then no gas in tank", would be represented as \( ('gas \; meter \; at \; zero', \; 'no \; gas \; in \; tank') \). We'll represent our beliefs as a set of strings in Python. (If you haven't seen the syntax for sets in Python, check out this documentation: https://docs.python.org/3/tutorial/datastructures.html#sets. Most syntax is the same as a dictionary with only keys, no values.)

We'll use a forward chaining system, in which the system repeatedly uses the rules to determine if anything new can be added to its beliefs. For instance, if the system only had the belief “gas meter at zero” and the rule “If gas meter at zero, then no gas in tank”, it could use forward chaining to add “no gas in tank” as an additional belief. The system is thus able to consider the logical consequences of its beliefs.

(a) Create a module `productionrules.py`. Write a function `get_triggered_rule(beliefs, rules)` that finds a rule that is “triggered” by a given set of beliefs. This function should take two parameters: (1) the beliefs (a set of strings), and (2) a list of rules, where each rule is a two-item tuple. Your function should return the first rule that allows the system to add something new to its beliefs. If no such rule exists, the function should return None.

For example, if the rule is “If a, then b” and the belief set contained a and b, then the rule is not triggered, because b is already part of the set of beliefs. If the belief set contained only b, then the rule is not triggered because the antecedent of the rule is not true. However, if the belief set contains only a, then the rule is triggered because this rule would allow us to conclude b, which we did not already know.

For this problem, you'll put your answers in `CS328-PS02-02.ipynb`. In the section `get_triggered_rule`, make a cell in that section which imports your module and tests that your `get_triggered_rule` function works properly.

(b) In forward chaining, we keep adding to our beliefs by executing the triggered rules until no more rules are triggered. “Executing a rule” simply means adding its consequent to our beliefs.

Here's an example: We start with the rules “If belief1, then belief2” and “If belief2, then belief3”, and the only belief in our belief set is “belief1”. We first execute rule 1, increasing our belief set to contain “belief1” and “belief2”. Then, with this expanded set of beliefs, rule 2 is trigger. We execute it so that our final set of beliefs is “belief1”, “belief2”, and “belief3”.

In your module, write a function `forward_chain(beliefs, rules)`, to automatically perform forward chaining using a given set of initial beliefs and list of rules. This function should return a tuple where the first item is the final set of beliefs and the second item is a list of rules that were triggered, in the order they were triggered. This function should use the function you wrote in the previous part, and should not modify the passed in beliefs or rules (i.e., it should be non-destructive).

In the section named `forward_chain`, include a cell to test that your function works properly.

(c) Imagine you have an old car whose battery often dies without warning. When your car battery dies you can’t charge your phone, turn on your headlights, or start the car. To make matters worse, you also happen to be quite forgetful and often neglect to fill up the gas tank when it is running low. When your gas tank is empty the gas meter on your dashboard reads zero and your car won’t start. We can represent this scenario in a production system as follows:

- If gas meter at zero then no gas in tank
- If no gas in tank then car won’t start
- If car battery is dead then head lights won’t turn on
- If car battery is dead then phone charger doesn’t work
- If car battery is dead then car won’t start

These rules are included in the section “Forward chain example 1”. In the cell with the rules, add code that uses your function(s) from above to run forward chaining given that you know that the car’s gas meter is at zero (and you start off knowing nothing else). Below this cell, write down what you can conclude about the states of the gas tank, head lights, phone charger, and car.

(d) Now, let's imagine a different production system that is similar but ultimately unrelated to the one we defined in the previous part:
If no gas in tank then gas meter at zero
If head lights won’t turn on then car battery is dead
If my phone charger doesn’t work then car battery is dead
If car won’t start then car battery is dead

These rules are in “Forward chain example 2”. In this new production system, if you knew only that your car won’t start what would you conclude? Intuitively, do your conclusion(s) seem like valid or reasonable inferences to make given the rules of your production system? Explain. Include code to support your conclusions if appropriate.

(e) Finally, we can imagine a third production system where the rules apply in both directions:

If gas meter at zero then no gas in tank
If no gas in tank then car won’t start
If car battery is dead then head lights won’t turn on
If car battery is dead then phone charger doesn’t work
If car battery is dead then car won’t start
If no gas in tank then gas meter at zero
If head lights won’t turn on then car battery is dead
If phone charger doesn’t work then car battery is dead
If car won’t start then car battery is dead

These rules are in “Forward chain example 3”. If you observed “no gas in tank”, what would you conclude? Does this conclusion seem reasonable given what you know about the meaning of the propositions? Briefly explain your reasoning. Include code to support your conclusions if appropriate.

(f) Do the parts above tell us anything about the limits of a basic production rule system for making inferences? In a section named “Limits of production rule systems”, explain how the system’s conclusions are consistent with or different from the types of conclusions that people make. Your answer should include how you think people would behave given a situation similar to the one in earlier parts of the questions, and should discuss the types of conclusions people might draw given particular evidence.

3. Another type of reasoning used in production role systems is backward chaining. In backward chaining, we still have beliefs and rules, but the basic method of inference is to begin with a goal proposition that we would like to prove, and then determine if we can prove that inference using our rules and beliefs. Some cognitive architectures, such as ICARUS\(^2\) actually have the ability to do both types of reasoning, which can be useful for making agents more efficient in particular kinds of situations.

In backward chaining, rules and beliefs take the same form as for forward chaining. Additionally, the goal is simply another logical proposition; as above, we’ll assume the statements contain no logical connectives, although the reasoning system can be applied in more complicated cases. The system first checks if its goal is one of its given beliefs. If it is, then the goal has been proven to be true. Otherwise, it finds any rules that have the goal as a consequent. For example, if we had the rules \([a \rightarrow b, b \rightarrow c, d \rightarrow c]\), belief \(a\), and goal \(c\), the system would find the rules \([b \rightarrow c, d \rightarrow c]\). It then uses the antecedent (left side) of these rules as new goals. In this example, it’s now trying to prove either \(b\) or \(d\). If repeats the process of checking if the (new) goal is one of its beliefs, and if not, finding rules that have the goal as a consequent. This process repeats until either there are no new rules to examine or a goal can be proven. In this example, the system would eventually have the goal of \(a\) (since \(a \rightarrow b\) and \(b \rightarrow c\)), and since this was a belief, it would recognize that it can prove the original goal \(c\).

(a) In your productionrules.py module, write a function \texttt{backward\_chain(beliefs, rules, goal)} that returns true if the given goal (a string) can be proven with the given beliefs and rules. The same representation of beliefs and rules is used as in the previous problem. As in the previous problem, your function should be non-destructive. In the section “backward\_chain” in CS328-P02-02.ipynb, include some tests cases in this section. Make sure to test a variety of cases, including rule systems that may have cycles and cases where the goal cannot be proven as well as cases where the goal can be proven.

(b) Create a new section “Comparing chaining systems”. In this section, consider whether one of backward or forward chaining is always more efficient if the system has a single proposition that it would like to evaluate as supported (proven true) or not by the current beliefs and rules. Explain your answer.

(c) Add a final section “Uses for chaining systems”, and explain whether it seems like it would ever be useful for an agent to have both reasoning systems available. You can assume that the agent is acting in the environment for a prolonged period of time and it may gain information over time.