CS 511: Artificial Intelligence II
Project 4: Reactive Learning Agent

`Agent Theseus`

Spring 2025

1 Concept

This project implements a **reactive learning agent (RLA)** that operates in the wumpus world. Specifically, it implements the agent function

$$f_{\mathrm{RLA}} : \Omega^* \to A$$

that maps the observed sequence of percepts to an action. The RLA operates in an **unknown** environment, in which the **forward probability** (the probability with which the agent goes forward on a `GO_FORWARD` action as opposed to slipping to the left or the right) is unknown. A forward probability of 1 means that the environment is completely deterministic. The forward probability is one of three values: 1, 0.8, or $\frac{1}{3}$, but the RLA doesn't know which *a priori*. The RLA spends some time collecting data through experience, from which it learns the forward probability using the **maximum likelihood estimation (MLE)**. This is the **exploration** phase. Once the forward probability is learnt, the RLA switches to the **exploitation** phase, where it uses the learnt forward probability along with the known transition model to navigate the environment and maximize its score. The RLA uses **condition-action rules** and a **deterministic model-based reflex strategy** if the learned forward probability is 1. Otherwise, if the learned forward probability is 0.8 or $\frac{1}{3}$, then the RLA has an option of using condition-action rules using either a **simple reflex strategy** or a **stochastic model-based reflex strategy**. In any case, the strategy used by the agent is **reactive**, hence its name. Another feature of the RLA is that if it learns the incorrect forward probability, it can sometimes **adaptively update** it to the correct value. I term this as the RLA being **selectively adaptive**. Section 3 discusses the learning algorithm in detail, and section 4 gives an overview of how exploitation is done using the different strategies.

2 Getting Started

The reactive learning agent is implemented in Scala. The implementation is contained in the files `src/scala/AgentFunctionImpl.scala`, `src/scala/ModelBasedReflexAgent.scala`, and `src/scala/ReactiveLearningAgent.scala`. The project directory contains a Makefile that automates building and running the RLA. The Makefile runs the project with the options `forwardProbability (-n)` set to 1 and `randomAgentLoc (-r)` set to `false`. It contains a `check` target that checks the system for the necessary tools (`scala`, `java`). It is recommended that the project is run after checking for the necessary tools as:-

```
$ make check
$ make #or "make run"
```

The above commands are for a single run by default. Of course, the `run` recipe can be updated with the `-t` option for multiple trials. A separate `make` target called `la-tenk` is provided for evaluating the RLA that runs 3,334 trials with a forward probability of 1, and 3,333 trials each with forward probabilities of 0.8 and 0.3334. It can be run using:-

```
$ make la-tenk
```

The score for each trial and the average scores for the three different modes are written to `wumpus_out.txt` or to the output file you specify using the `-f` option in the recipe.

The project was tested using:-

- **Scala Version:** 3.7.0
- **Java Version:** OpenJDK 22.0.1

3 Learning

The idea for learning is that the agent faces the south wall in $(1,1)$ when it starts, and keeps going forward i.e. `bump`ing into the wall for 15 time steps or until **two** *failures* are encountered. A *failure* is encountered when the agent expects a `bump` percept because it is going forward into the wall, but doesn't get one because the `GO_FORWARD` action is stochastic. On the first failure, it can be inferred with certainty that the agent has slipped to the left into $(2,1)$, since going forward or slipping to the right would have resulted in a `bump`. The boolean values of the `bump` percepts throughout the learning phase are collected in a boolean vector called the **learning experience**. Hence, a `false` value in the learning experience corresponds to a failure. Since the agent can encounter at most two failures during the learning phase, the learning experience has at most two `false` values, and the rest are `true`. Moreover, if less than two failures have been encountered, then the size of the learning experience is 15. The agent learns the forward probability using a maximum likelihood estimate from the data in the learning experience:

1. If the agent encounters 0 failures then it learns the forward probability to be **1**.
2. If the agent encounters 1 failure then the forward probability is certainly not 1. The maximum likelihood estimate is **0.8**, which is what the agent learns.
3. If the agent encounters 2 failures then, like the 1 failure case, the forward probability is certainly not 1. In this case, however, it is not intuitively clear which value out of 0.8 or $\frac{1}{3}$ should be learned. Instead of an arbitrary (and most likely suboptimal) choice, the agent follows a rigorous maximum likelihood estimation (MLE) procedure. We know that the learning experience looks like

$$\underbrace{TT\cdots T}_{n}F\underbrace{TT\cdots T}_{m}F$$

where the first n `true` values are successes in $(1,1)$, the first `false` value is the failure encountered in $(1,1)$ that takes the agent to $(2,1)$, the following m `true` values are the successes in $(2,1)$, and the final `false` value is the failure in $(2,1)$ that takes the agent either to $(1,1)$ or $(3,1)$. If the forward probability is p then the likelihood of obtaining this learning experience, L, is

$$P(L;p) = \left(\frac{1+p}{2}\right)^n \left(\frac{1-p}{2}\right) p^m (1-p)$$

In maximum likelihood estimation, the objective is to maximize the likelihood $P(L;p)$, which is equivalent to maximizing the **log-likelihood**:

$$\log P = n\log\left(\frac{1+p}{2}\right) + \log\left(\frac{1-p}{2}\right) + m\log p + \log(1-p)$$

To maximize the log-likelihood, I equate its derivative with respect to p to 0:

$$\begin{aligned}\frac{\partial \log P}{\partial p} &= \frac{n}{1+p} - \frac{1}{1-p} + \frac{m}{p} - \frac{1}{1-p}\\ &= \frac{n}{1+p} + \frac{m}{p} - \frac{2}{1-p} = 0.\end{aligned}$$

Multiplying through by the common denominator $p(1+p)(1-p)$ gives:

$$np(1-p) + m(1+p)(1-p) - 2p(1+p) = 0.$$

Expanding and simplifying, I obtain:

$$-p^2(n+m+2) + p(n-2) + m = 0,$$

or equivalently,

$$p^2(n+m+2) - p(n-2) - m = 0.$$

Applying the quadratic formula with the fact that forward probability estimate must be positive, the maximum likelihood estimator is then

$$\begin{aligned}
\hat{p}_{\text{MLE}} &= \frac{(n-2) + \sqrt{(n-2)^2 + 4m(n+m+2)}}{2(n+m+2)} \\
&= \frac{(n-2) + \sqrt{n^2 - 4n + 4 + 4mn + 4m^2 + 8m}}{2(n+m+2)} \\
&= \frac{(n-2) + \sqrt{(n^2 + 4mn + 4m^2) - (4n - 8m) + 4}}{2(n+m+2)} \\
&= \frac{(n-2) + \sqrt{(n+2m)^2 - 4(n-2m) + 4}}{2(n+m+2)}.
\end{aligned}$$

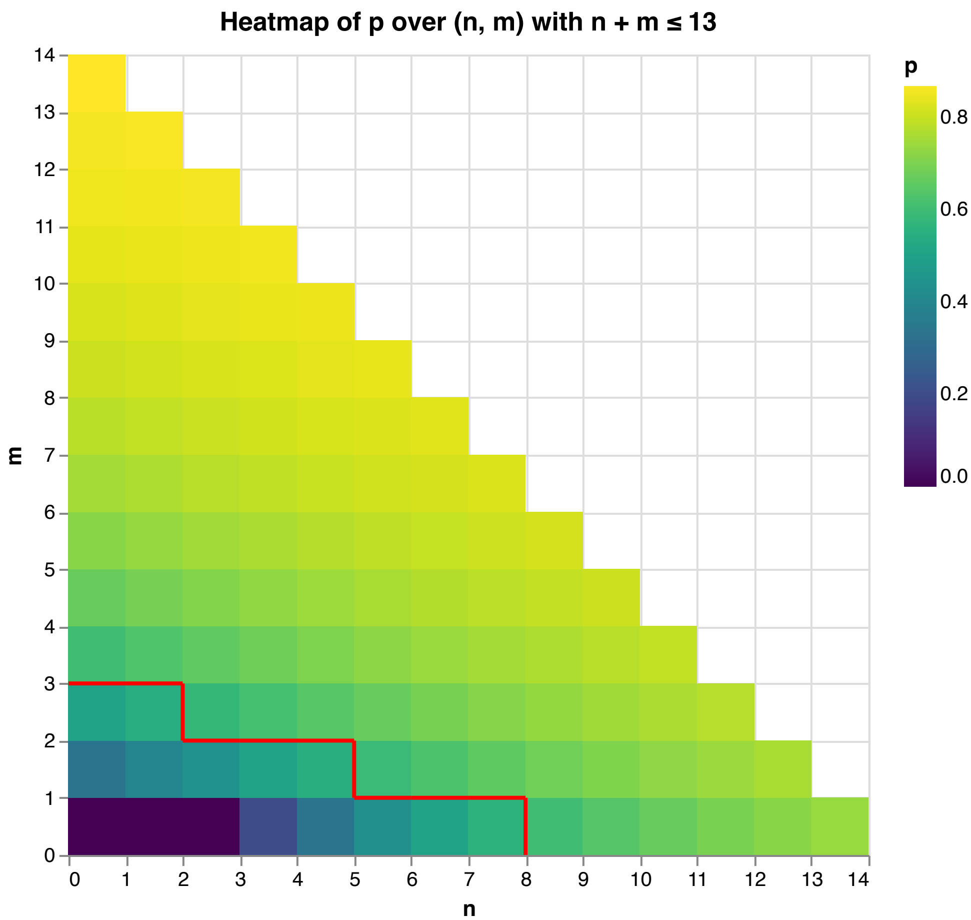
Adding and subtracting $4(n+2m)$ from the discriminant,

$$\begin{aligned}
\hat{p}_{\text{MLE}} &= \frac{(n-2) + \sqrt{(n+2m)^2 + 4(n+2m) - 4(n+2m) - 4(n-2m) + 4}}{2(n+m+2)} \\
&= \frac{(n-2) + \sqrt{((n+2m)^2 + 4(n+2m) + 4) - 4(n+2m) - 4(n-2m)}}{2(n+m+2)} \\
&= \frac{(n-2) + \sqrt{(n+2m+2)^2 - 8n}}{2(n+m+2)}.
\end{aligned}$$

$\therefore$ I have got the MLE estimate in nice form. The maximum likelihood estimate in this case is

$$\boxed{\hat{p}_{\text{MLE}} = \frac{(n-2) + \sqrt{(n+2m+2)^2 - 8n}}{2(n+m+2)}}$$

The agent computes this based on the learning experience exactly as shown and learns the forward probability that is closer to $\hat{p}_{\text{MLE}}$ out of 0.8 and $\frac{1}{3}$.



The heatmap of $\hat{p}_{\text{MLE}}$ over the (n, m) space includes a boundary in red within which p is learnt as $\frac{1}{3}$, and outside of which, p is learned as 0.8. As one can see, the number of (n, m) pairs for which $\frac{1}{3}$ is learnt is much lesser than the number for which $p = 0.8$ is learnt.

3.1 Cases of `breeze` and `stench`

If the agent observes a `breeze` during the learning phase, then it immediately gives up i.e. chooses `NO_OP` for every subsequent action. This is because in such a case, the `breeze` is observed either in (1, 1) or (2, 1). If the `breeze` is observed in $(1, 1)$, then it is optimal to give up and accept a 0 score as discussed for the model-based reflex agent. If the `breeze` is observed in $(2, 1)$ during the learning phase, then it is too risky to even continue learning as the agent might slip into $(3, 1)$ and die.

If the agent observes a `stench` during the learning phase, then it `SHOOT`s along the $x = 1$ row to get the wumpus out of the way while learning. This becomes doubly beneficial as even if the agent misses (doesn't observe a `scream`), the position of the wumpus becomes known with certainty.

4 Exploitation

After learning (exploration), the next phase is exploitation where the agent uses its learnt knowledge to navigate the environment and try and maximize its score. The agent uses different exploitation strategies depending on the forward probability learnt.

4.1 Deterministic Model-Based Reflex Strategy ($\hat{p}_{\mathbf{MLE}} = 1$)

In the deterministic case, the RLA simply delegates the processing to a model-based reflex agent (MRA) architecture, which was implemented in Project 2 for the deterministic case.

4.2 Simple Reflex Strategy ($\hat{p}_{\mathbf{MLE}} = 0.8 \quad \textbf{or} \quad \frac{1}{3}$)

The simple reflex strategy for the stochastic case optimizes for the worst case. The agent keeps executing the `GO_FORWARD` action until the slightest possibility of death on another `GO_FORWARD`, after which it gives up (keeps `NO_OP`ing). The give up condition includes the first observation of a `breeze` or the first observation of a `stench` after having shot already.

In this strategy, the agent relies on the nondeterminism of the `GO_FORWARD` action to translate it sideways for exploration, and itself never executes a turning action. The agent never dies in this strategy given that the forward probability is learnt correctly.

4.3 Stochastic Model-Based Reflex Strategy ($\hat{p}_{\mathbf{MLE}} = 0.8 \quad \textbf{or} \quad \frac{1}{3}$)

In the stochastic case, the RLA maintains a full belief state of the environment along with probabilities. The RLA executes Bayesian updates of the belief state based on actions and observations. At each time step, the RLA computes the probability of death and the probability of finding the gold for moving in all four directions. The RLA chooses the direction to move for which the probability of gold exceeds the probability of death by the maximum difference. If the probability of death exceeds the probability of gold for all directions then the agent simply `NO_OP`s and hence gives up. In case the agent observes a `stench`, the agent runs the same algorithm with an intermediate `SHOOT` action in all directions. This strategy enables the agent to find plans such as continuously bumping into a wall in a dangerous zone, and relying on the nondeterminism of the `GO_FORWARD` action to get it out of the tight spot. This strategy is not as optimized for the worst case as the simple reflex strategy since then agent would disregard a good chance of death if the chance of gold is higher.

4.4 Selective Adaptiveness ($\hat{p}_{\mathbf{MLE}} = 1 \to \hat{p}_{\mathbf{MLE}} = 0.8$)

In some cases, the agent incorrectly learns $\hat{p}_{\mathrm{MLE}} = 1$ in an environment with $p = 0.8$ since it never slips or *fails* during the learning phase. The agent exploits this learnt knowledge with a deterministic model-based reflex strategy, for which it delegates the processing to the model-based reflex agent architecture from Project 2 as mentioned before. By properties of that MRA architecture, the agent should never observe a `bump` if the environment is truly deterministic. However, in the case where it does (which is possible since $p = 0.8$ was learnt incorrectly as 1), the agent realizes that it has learnt the forward probability incorrectly, and immediately switches to the stochastic model-based reflex strategy from the deterministic one. Hence, the agent **adapts** its learnt estimate according to the observation even in the exploitation phase after the learning (exploration) phase. However, such an adaptation occurs only when $p = 0.8$ is incorrectly learnt as $\hat{p}_{\mathrm{MLE}} = 1$, which the agent realizes when it observes a `bump` in the exploitation phase. Any other incorrectly learnt estimates go unnoticed and more often than not result in death. This makes the adaptiveness selective.

5 Results

The reactive learning agent uses a deterministic model-based reflex strategy for $\hat{p}_{\mathrm{MLE}} = 1$, a simple reflex strategy for $\hat{p}_{\mathrm{MLE}} = 0.8$, and a stochastic model-based reflex strategy for $\hat{p}_{\mathrm{MLE}} = \frac{1}{3}$.

The RLA was evaluated using 3,334 trials with $p = 1$, 3,333 trials with $p = 0.8$, and 3,333 trials with $p = \frac{1}{3}$, making a total of 10,000 trials.

Results for the $p = 1$ case (with 3,334 trials):

Minimum	1st Quartile	Median	Mean	3rd Quartile	Maximum	Std. dev.	Mode
-1058	0	948	496.8047	970	1000	542.6085	0

Results for the $p = 0.8$ case (with 3,333 trials):

Minimum	1st Quartile	Median	Mean	3rd Quartile	Maximum	Std. dev.	Mode
-1056	-19	-6	170.813	0	1000	438.961	0

Results for the $p = \frac{1}{3}$ case (with 3,333 trials):

Minimum	1st Quartile	Median	Mean	3rd Quartile	Maximum	Std. dev.	Mode
-1040	-12	-3	185.3213	0	1000	404.2231	0

$\therefore$ The reactive learning agent was run 10,000 times and an average score of **284.3403** was achieved. The following are the summary statistics:-

Minimum	1st Quartile	Median	Mean	3rd Quartile	Maximum	Std. dev.	Mode
-1058	-14	0	284.3403	960	1000	489.3	0

6 Reflection

- The estimator is wrong! It assumes that the parameter p is continuous on $[0, 1]$ when in reality it's discrete with possible values $\frac{1}{3}$ or 0.8. You can instead do a Bayesian inference since it's known that the prior on p is uniform. Hence, take it to be a random variable P. The optimal estimator is the following: we have obtained the conditional PMF $P_{L|P}(\ell \mid p)$, where ℓ is the observed learning experience parameterized by n and m, and it would suffice to return

  $$p^* = \arg \max_{p \in \{\frac{1}{3}, 0.8\}} P_{L|P}(\ell \mid p)\,.$$

  The reason this works is since the prior is uniform, $P_{P|L}(p \mid \ell) \propto P_{L|P}(\ell \mid p)$. Estimating a continuous p and snapping it to the nearest possible value doesn't necessarily give the optimal.

- The RLA code could've been better documented and better designed (less smelly). But I was short on time.