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

`Agent Theseus511`

Spring 2025

1 Concept

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

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

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 estimator (MLE)**. This is the **exploration** phase. Once the forward probability is learnt, the RLA switches to the **exploitation** phase, where it uses the learned forward probability along with the known transition model to navigate the environment and maximize its score. The RLA uses **condition-action rules** and a **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 process in detail, and section 4 gives an overview of how exploitation is done using the different strategies.

2 Getting Started

In the submission, the RLA uses condition-action rules with a stochastic model-based reflex strategy if a forward probability of 0.8 or $\frac{1}{3}$ is learned. However, switching to a simple reflex strategy in the stochastic case with some control is as simple as uncommenting two lines of code and commenting out two others. However, the stochastic model-based reflex strategy performs better.

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:** 3.6.3
- **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

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

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, $\mathcal{L}$, is

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

In maximum likelihood estimation, the objective is to maximize the likelihood function $L(\mathcal{L} \mid p)$, which is equivalent to maximizing the **log-likelihood**:

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

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

$$\frac{\partial \log L}{\partial p} = \frac{n}{1+p} - \frac{1}{1-p} + \frac{m}{p} - \frac{1}{1-p}$$
$$= \frac{n}{1+p} + \frac{m}{p} - \frac{2}{1-p} = 0.$$

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

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

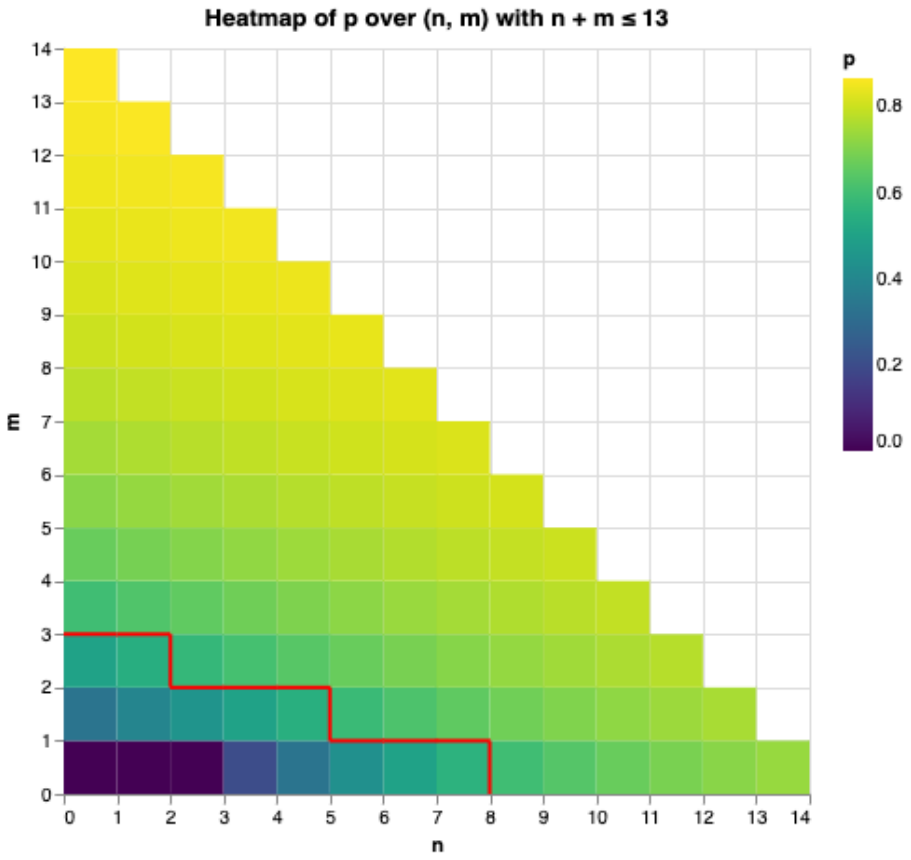
Adding and subtracting $4(n+2m)$ from the discriminant,

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

$\therefore$ 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 remove any hindrances in the learning process. 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.

5 Results