Norwegian University of Science and Technology
IT3708 – Bio-Inspired Artificial Intelligence

Project 1: Supervised and Reinforcement Learning of Neural Agent Controllers

Per Magnus Veierland
permve@stud.ntnu.no

January 20, 2017

Implementation and Baseline

The assignment program is a single file `flatland.py` written using the Python programming language with the NumPy library for computation. PyQt bindings are used to render vector graphics to PDF. The visualization uses an X to mark the agent starting point, and an increasing line thickness to indicate the direction of movement. The most important classes are the `BaselineAgent` and `LearningAgent`. They both implement the method `act` which takes a `percepts` parameter and returns an `Action`. The `percepts` parameter is a $(3, S)$ N-dimensional array, where the first dimension is the direction, and the second is the sensor range (S). The `LearningAgent` also provides the methods `evaluate`; which takes a `percepts` parameter and returns a tuple of one-hot encoded inputs and the neural network outputs, and the method `update_weights`; which updates the network weights given a learning rate, a delta, and network inputs. `SupervisedAgent` and `ReinforcementAgent` both inherit from `LearningAgent` and implement separate `train` methods.

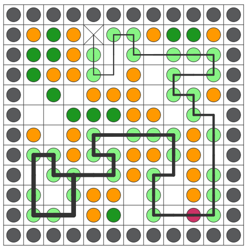

Figure 1: Flatland visualization of baseline agent

The baseline agent behavior is to always move towards food. If there is no neighboring food, it will move towards an empty location. If there are no empty locations it will move towards poison. Given how the world is defined the baseline agent will never need to move into walls. If there are more than one possible destination of the same kind, it will prefer moving forward, otherwise left, otherwise right, accordingly.

The mean score achieved by the baseline agent is 20.2 averaged over 1 million trials.

Supervised Learning Agent

The computation of the delta term as part of the *Widroff-Hoff rule* is shown in Figure 2. The *CorrectChoice* is a *one-hot encoded* vector where the index of the target action is set to 1. To make the softmax computation numerically stable the reformulation described in Equation 1 is used, with $\log C = -\max_k y_k$, such that the greatest exponent factor is 0.

$$\frac{e^{y_j}}{\sum_k e^{y_k}} = \frac{C \cdot e^{y_j}}{C \cdot \sum_k e^{y_k}} = \frac{e^{y_j + \log C}}{\sum_k e^{y_k + \log C}} \tag{1}$$

```
def train(self, percepts, target_action, learning_rate):
  inputs, outputs = self.evaluate(percepts)
  outputs        -= np.max(outputs)
  softmax         = np.exp(outputs) / np.sum(np.exp(outputs))
  correct_choice  = encode_int_as_one_hot(target_action, 3)
  delta           = correct_choice - softmax
  self.update_weights(learning_rate, delta, inputs)
```

Figure 2: The `train` method of `SupervisedAgent`.

Reinforcement Learning Agent

The computation of the delta term for the agent trained using reinforcement learning is shown in Figure 3. The network is evaluated using the `percepts` from both before and after the action is applied. The current *Q-value* is the maximum network output given the pre-action `percepts`, and the next *Q-value* is the maximum network output given the post-action `percepts`. Since only the neuron corresponding to the taken action is updated, the scalar $r + \gamma \max_{a'} Q(s', a') - Q(s, a)$ is multiplied with a *one-hot encoded* action vector.

Analysis

A clear structure can be seen in the reinforcement agent weights (Table 2), where for each direction the four weights linked to the input to the output is ordered in the same way such that the “food weight” is the strongest, then the “empty weight”, then the “poison weight”, then the “wall weight”.

The performance of the agent trained using supervised learning closely matches the performance of the baseline

Action	LE	LW	LF	LP	FE	FW	FF	FP	RE	RW	RF	RP
Left	2.27238	-1.99947	3.55820	-1.56634	0.74275	0.06020	0.65737	0.80553	0.45891	0.59558	0.53390	0.67514
Forward	0.59295	0.32543	0.56372	0.66674	2.55852	-2.98390	3.89967	-1.32605	0.68196	0.03736	0.65998	0.77091
Right	0.50691	0.67649	0.26464	0.51774	0.50082	0.35558	0.41975	0.68855	2.34135	-2.97435	3.87039	-1.27376

Table 1: Weights for agent trained using reinforcement learning with 30 training rounds of 100 Flatland worlds each ($S = 1$).

Scenario	Network Inputs			Network Outputs			Action	Description
	Left	Forward	Right	Left	Forward	Right		
1	Wall	Poison	Food	−0.66004	−0.34063	5.23544	Right	The network expectedly shows the least preference for the wall, higher preference for the poison, and a greatly higher preference for the food.
2	Wall	Poison	Wall	−0.59836	−0.96325	−1.60930	Left	This is a scenario never encountered during training, and the agent makes a move towards a wall when it should have moved towards poison.
3	Empty	Empty	Wall	3.61071	3.18884	−1.96661	Left	In this scenario, the agent moves away from the wall instead of moving forward. This is a better action as it increases the probability of avoiding poison in the next turn.
4	Wall	Empty	Empty	−0.79781	3.56591	3.51867	Forward	This scenario mirrors scenario #3, however the agent moves forward instead of to the side. It can be seen that the two action outputs are very similar, and this would likely be corrected with further training.
5	Wall	Food	Food	−0.80820	4.88509	4.96663	Right	In this scenario, the agent moves away from the wall into the food instead of moving forward into the food.

Table 2: Selected scenarios illustrating agent behavior resulting from reinforcement learning ($S = 1$).

agent. This was expected as the baseline behavior is a simple function of preferring food over empty areas, empty areas over poison, and poison over walls. It is also expected that the supervised agent would not surpass the baseline agent since it should learn the baseline agent function. When comparing the reinforcement agent with sensor range 1 to the baseline agent it performs slightly, but distinctly better. This was unexpected as the baseline agent was designed to be optimal.

To understand why the reinforcement agent performs slightly better the scenarios are looked at more closely. With a sensor range of 1 there are 64 possible inputs scenarios, out of which 44 are obvious (e.g. F/W/W), 6 are left/right ambiguous (e.g. F/W/F), 6 are sideways/forwards ambiguous (e.g. F/F/O), 4 are left/forwards/right ambiguous (e.g. F/F/F), and 4 are non-obvious since they involve a wall on a side and food or an empty area (e.g. W/F/O. If the behavior of the baseline agent is adjusted such that in scenario #5 it moves to the side instead of forwards, it ends up performing as good as the reinforcement trained agents, with a score of 20.8 when averaged over 1 million trials. This is an improvement as moving away from the wall increases the probability of avoiding poison. Any slight remaining discrepancy between the reinforcement agent and baseline agent scores is hypothesized to be caused by the reinforcement agent not moving in the same direction for all left/right ambiguous situations, but balancing between both directions for different scenarios.

An agent trained using reinforcement learning and a sensor range of 3 expectedly performs significantly better than the other agents as it is able to exploit more information, e.g. preferring a direction which has two food entities in a row, and avoiding poison or walls which are more than one move away.

```
def train(self, percepts, percepts_next, learning_rate,
          discount_factor, reward):
  inputs, outputs = self.evaluate(percepts)
  action = np.argmax(outputs)
  q_current = outputs[action]
  q_next = np.amax(self.evaluate(percepts_next)[1])
  delta  = (encode_int_as_one_hot(action, 3) *
    (reward + discount_factor * q_next - q_current))
  self.update_weights(learning_rate, delta, inputs)
```

Figure 3: The `train` method of `ReinforcementAgent`.

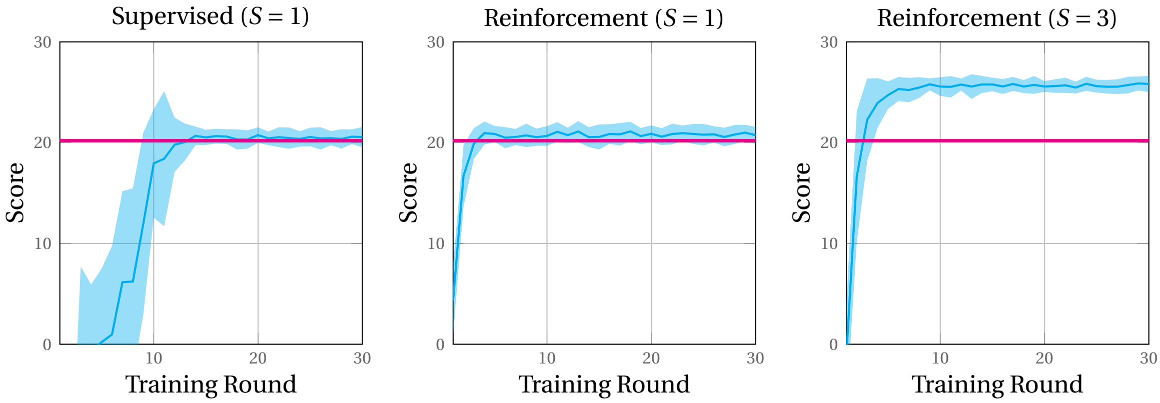

Figure 4: Agent scores showing gradual learning. Each training round consists of 100 random Flatland worlds. The dark blue lines show the mean across 30 repetitions and the light blue shows the standard deviation. The thick magenta line shows the baseline agent score (20.2). A learning rate $\alpha = 0.01$ and discount factor $\gamma = 0.9$ is used for all experiments.