A Reinforcement Learning Approach for Smart Farming

Gabriela ENE
The Bucharest University of Economic Studies, Romania
gabriela.ene02@gmail.com

At a basic level, the aim of machine learning is to develop solutions for real-life engineering problems and to enhance the performance of different computers tasks in order to obtain an algorithm that is highly independent of human intervention. The main lying ingredient for all of these, is, of course, data.
Data is only valuable if it is transformed into knowledge, or, experience and the machine learning algorithm is only useful if it can make a prediction with high accuracy outside the examples in the training set. The field of machine learning intersects multiple domains such as data science, artificial intelligence, statistics, and computer science, but has appliances in any possible field that relies on decision making based on evidence, including healthcare, finance, manufacturing, education, marketing and recently, more and more in agriculture and farm-related management systems. As the Internet of Things and Cloud-Based solutions are introducing artificial intelligence in farming, the phenomenon of Big Data is going to impact the whole food-supply network. Machines that are connected with each other through a network or that are equipped with deep learning software or just with measurement systems are making the farming processes extremely data-driven. Fast decision-making capabilities might become a game-changing business model in this field.
Keywords: *machine learning, reinforcement learning, artificial intelligence, smart farming, Thompson Sampling, Q-Learning*

1 Introduction

Precision agriculture — a suite of information technologies used as management tools in agricultural production— has already advanced and will continue to change farm management, from the way farmers consider their commodity mix, scout fields, and purchase inputs, to how they apply conservation techniques, and even how they price their crops and evaluate the long-run size of their operations [1]. Mainly, the main focus of researchers and one big improvement for the farmers is the analytical causality between seeds and fertilizers or be-tween irrigations and crop quality. Traditional methods to determine relationships between such inputs and outputs relied on experiments or estimating data by mixing observed data sets with behavioral models, such as two-stage least square technique.
For a vast majority of farmers, the small plot experiments are mainly focused on few inputs and restricted to a determined time/season/location and cannot be often generalized, so the results may not be relevant and such implementation might be costly. The intersection of machine learning and agriculture might offer the starting point of a broader solution de-signed to optimize crop management. The aim of this article is to cover the implementation and the impact of reinforcement learning algorithms in smart farming, starting from the problem that many farmers face when they choose between using past production methods that bring income and exploring the value of new practices that can increase income. This problem fits under exploration vs. exploitation paradigm, and the focus of this paper is to conceptualize it as a multi-armed bandit problem. Also, on the same note, considering the increased costs of transportation, a conceptual implementation of a self-driven truck in an established environment is presented.

2 Content details

2.1 Types of machine learning algorithms

Machine learning algorithms can be classified into three categories:

- Supervised Learning
- Unsupervised Learning
- Reinforcement Learning

First-class needs a labeled data set in order to acquire the optimal knowledge. One good example would be the classification of a never-before-seen item based on a trained model that includes many items recorded in the data set with a corresponding label. Saying that z is the feature vector, as, in the data instance, the equivalent label of z would be f(z), known as the ground truth of z. The feature vector can be a multi-dimensional vector of different features that are relevant to the item, and the value of f(z) is one of a couple of classes of which the item belongs, so the model, is basically a classifier. If f(z) can have multiple values and the outcome values are ordered, then the model is a regressor.
Considering a prediction model of z as p(z), the success of the model under the influence of a parameter, $p(z|\theta)$, depends on the distance between these two vectors: f(z) and $p(z|\theta)$. This distance is known as the cost. The main goal of supervised learning is to minimize the cost, so to determine the parameters of the model that among all the data points of z, result in minimum cost.

Unsupervised learning assumes modelling data without knowing the associated labels.
Dimensionality reduction and clustering are very powerful tools that are broadly used to gain knowledge from data alone. The first one implies removing redundant features, in order to lower the dimensional space of the feature vectors, and clustering manages the process of distributing the data in specific classes without considering the pre-defined labels.
Unlike training labeled datasets provided by an external „teacher", and different from the approach of finding patterns in unlabeled datasets, reinforcement learning challenges the trade-off between exploration and exploitation.

2.2 Reinforcement learning

The very basic definition of reinforcement learning is acquiring knowledge through interaction with an environment. An agent acts in a specified environment and adapts its behaviour based on the rewards that it receives. The roots of the trial-and-error process are in behaviourist psychology [2], the agent main goal being to learn a strategy *[policy]* that would maximise the cumulative reward.
Reinforcement learning theory is already contributing to our understanding of natural reward, motivation, and decision-making systems, and it can contribute to the improvment of human abilities to learn, to remain motivated, and to make decisions [3].
The agent in the reinforcement learning algorithm, at a predefined time step, t, detects a state, s_t. The interaction with the environment assumes taking an action a_t, that will trigger the transition of both the agent and the environment to a new state s_{t+1}, defined by the previous one and the taken action. The state consists of sufficient statistics in order to offer the agent all the needed data in order to proceed in the best direction.
The rewards given by the environment determine the optimal sequence of actions, formally called „*policy*". The change of the state consists also in providing feedback to the agent, as a scalar reward r_{t+1}. Knowing the state, the policy will return a single action or a set of actions to perform.
One efficient technique to describe the

environment in an RL problem would be the *Markov Decision Process* approach, which provides an efficient model that can perform probabilistic inference over time. [4]

Markov Decision Process elements are as follows:

- The set of states - *S*
- The set of actions - *A*

Each s(i) state has its corresponding action or set of actions A(s(i)).

- The transition probabilities model P $\{S_{t+1} = s \mid St = s, A_t = a\}$

The probability of going from state s_t to state s_{t+1} depends only on the action and on the state.

- Reward function - *R(s)*
- Discount factor: $\gamma \in [0, 1)$

Once the agent takes an action a_t, selected from a set of actions that correspond to the state s_t, the agent gets the expected value of the reward, *R(s,a)* and, given the transition probability, the state of the process moves to the next one, so the model builds a path of transited states. Policy π is a mapping from states to a probability distributions over actions $\pi(s,a)$. So, it describes the way of acting. The function depends on the action and the state and returns the probability of taking the action in the specific state.

$$\sum_{a} \pi(s, a) = 1$$

The scope of the RL is to get the maximum reward from all states, with the optimal policy:

$$\pi^* = \frac{\operatorname{argmax}_\pi \mathrm{E}[R|\pi]}{\pi}$$

Learning the optimal policy implies using one of the two types of value functions available in machine learning: an action-value function - *Q(s,a)* – or a value function *V(s).*

Following a policy in state *s*, the expected return would be given by the formula:

$$V^{\pi}(s) = \mathrm{E}_{\pi}[R_t|s_t = s]$$

Even though the state is the same, the value function varies depending on the policy. The action-value function returns the value added by taking an action in a specified state when approaching some policy.

$$Q^{\pi}(s, a) = \mathrm{E}_{\pi}[R_t|s_t = s, a_t = a]$$

We can rewrite the value function in this manner:

$$V^{\pi}(s) = \mathrm{E}_{\pi}[r_{t+1} + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots|s_t = s] = \mathrm{E}_{\pi}[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|s_t = s]$$

Taking into account the transition probability, and the expected reward that the agent receives by taking the action *a* and moving to state $s_{t+1,}$ we obtain the Bellman equation for the action value function:

$$V^{\pi}(s) = \sum_a \pi(s, a) \sum_{s'} P^a_{ss'} [R^a_{ss'} + \gamma V^{\pi}(s')]$$

And also, we can do this for the action-value function:

$$Q^{\pi}(s) = \sum_{s'} P^a_{ss'} \ [R^a_{ss'} + \gamma \sum_{a'} \pi(s', a') Q^{\pi}(s', a')$$

This equation is important because allows expressing values of one state as values of another state, so if we know the value of a specific state we can determine the value of another.

3. The Thompson Sampling Algorithm

Thompson sampling is an algorithm for online decision problems where actions are taken sequentially in a manner that must balance between exploiting what is known to maximize immediate performance and investing to accumulate new information that may improve future performance [5].

Although it was first proposed in 1933, it is only in the past years that interest into its potential developed, and currently, it has been successfully applied in a broad variety of domains, especially in website management, A/B testing, portfolio management, or recommendation systems. The concept of the n-armed bandit problem is as follows: among a set on n actions, the agent is asked to make a choice.
Every choice will be rewarded with a numerical value selected from a stationary probability distribution. The objective of the agent is to maximize the value received after each action over a number of fixed iterations, or time steps. The greedy approach of this assumes selecting the action that will return the highest reward, a phase that corresponds to the exploitation part. Improving the estimate of a reward, by choosing one of the nongreedy actions is the exploration phase.
The multi-armed bandit problem is often presented as a slot machine with *n* arms. By pulling one arm at a time step, a reward is given and over M number of rounds, the player's scope is to obtain the maximum sum of the rewards.
Given the fact that the rewards are random, each one of the *n* arms defines for k ∈ {1, 2, … , n} a stochastic process $\{X_{i,m}\}$ in the form of a distributed sequence of random variables, with an unknown mean μ_i. One specific type of the bandit problem is the *Bernoulli Bandit*, which models the probability of an event occurrence, that follows a binomial distribution, with N = 1, which is basically, the Bernoulli distribution.
This model can be adapted to the problem of the farmer that needs to decide which plots to select as experimental plots for different seeding rates.
Supposing that the farmer manages many fields, the purpose of our model is to decide where to place experimental plots in order to obtain improved yield response.
Each field has different soil characteristics, such as nutrients, acidity/alkalinity, organic matter or type.
The decision implies using all this information for better placement of the experimental plots in the field. The feature vector, *v*, of a field consists of a predefined number of similar characteristics for each field and an area in the fields is defined by $\mathbf{M} = \sum_i |\mathbf{v(i)}|$, the total of the feature values. Let's assume that *v(1)* describes parts of the field by nutrients content, with values varying in different ranges: less than 4%, between 4.5% and 5%, 5% and 7% and over 7%, so the field would be divided into four areas.

Following the same approach, *v(2)* can classify the field into five areas depending on the pH value, so we would have 9 parts of the field that can overlap. Coming back to our model, each of these parts nine parts is an "arm' of the multi-armed bandit problem. By selecting an area and placing a plot there, the farmer observes and figures if the plot improved the total reward, in this case, the yield response.

The model we follow to track the yield response to the seeding rate is as follows, as propesed in [6] :

$$\text{Yield} = Ymax \times (1 - e^{-\beta \times SR}) \text{ [6]}$$

Ymax is the estimated asymptotic yield maximum, and β determines the responsiveness of yield as seeding rate increases.

Therefore, a smaller β indicates that a higher seeding rate is needed to reach maximum yield for that seed treatment. [6]

The nonlinear least squares (NLS) was used to estimate the parameters *Ymax* and β separately, an estimation that can be achieved by the algorithm [6].

Each area from the fields is assigned, at each step, a probability that selecting a plot that belongs to it will improve the estimation of the parameters.
The probability function is based on the previous steps, which resulted in the better or worse estimation of the parameters.

A reward of *1* is added if selecting the field area improved the accuracy of the prediction, and *0* otherwise. After that, the area with the greatest probability of improving the estimation is selected. At each iteration of sample selection, a new sample will be added to the training dataset.
The expected rewards are modeled using a probability that follows Bernoulli distribution with parameter $\pi_i \in [0, 1]$. We maintain an estimate of the likelihood of each πi given the number of successes αi and failures βi observed for the field area. Successes (r = 1) and failures (r = 0) are defined based on the reward of the current iteration. It can be shown that this likelihood follows the conjugate distribution of a Bernoulli law, a Beta distribution *Beta(αi , βi)* [7] :

$$P(\pi i \,|\alpha i, \beta i) = \frac{\Gamma(\alpha i + \beta i)}{\Gamma(\alpha i)\Gamma(\beta i)} (1 - \pi i)^{\beta i - 1} \times \pi i^{\alpha i - 1}$$

Thompson Sampling for Sample Selection [7]

1: Ymaxi = 1, βi = 1,S={}, M={areas}, A={1,2,…M},N=field areas ,∀i ∈ {1, . . . , M}
2: for t = 1, , ...N do
3: for i 1, . . . , N do
4: Daw ˆπi from Beta(Ymaxi, βi)
5: Reveal sample ht = {xt, yt, mt} from field areas Cj
where j := arg maxi ˆπi.
6: Add sample ht to S and remove from all field areas.
7: Obtain new model parameters Ymaxi, βi
8: Compute reward rt based on new prediction:

$$\text{Yield} = \text{Ymax} \times (1 - e^{-\beta \times SR})$$

9: if rt == 1 then Ymaxj = Ymaxj + 1
10: else βj = βj + 1

4 Q-Learning Algorithm

The underlying philosophy of this algorithm is based on the following method: the agent takes an action at a particular state and the feedback consists of a reward or a penalty. The agent can evaluate the feedback by estimating the value of the state to which it was taken. So, the learning is the process of going through different stages with the scope of maximizing the future return, R. The return from a specific time step, r_t, can be defined also by using the discount factor, γ, where $0 < \gamma < 1$, defined before as an element in the Markov Decision Process. The important thing to consider is that if the value of this factor is smaller, the agent would be inclined to choose only the immediate reward and not take into consideration the up-coming rewards.
If $\gamma = 1$, then all rewards are equally considered. The algorithm makes use of the action-value function and estimates the optimal function, with no regards of the policy that it follows. But, the policy is used also in this approach in order to map the pairs of states and actions that were updated.

The applications of this algorithm in farming are many, but we will consider one simulation, a delivery truck that is self-driven.
The truck's job is to get the crop from a determined place and to deliver it to another. Basically, the reinforcement algorithm that we will model will follow pre-defined steps as: environment observation, deciding upon the action to take based on the strategy of maximizing the obtained reward, acting, receiving the penalty or the reward, accumulating experience and improving the strategy and, finally, iterating until the optimal function is found.
A high positive reward is going to be obtained for a successful arrival at the location, and a penalty will be given if the truck arrives in the wrong place. The discount factor will be used when not getting to the destination after every step, meaning that late-arriving is better than making wrong moves.
The state-space consists of all situations that the truck may encounter and consists of useful data needed in the decision-making process.
Assuming that the field is the training area of the truck, we don't have to consider many obstacles that might be encountered, but only the area of the field, which we can divide in small plots, viewed as a matrix M.
For the purpose of the example, we would consider 36 possible plots, some of them may contain the silo and some can contain the harvested crop.
The actions will be defined as crop-load, crop-delivery, west, north, east and south. In the code, we would assign a penalty for every stop at the wrong silo location. The algorithm will only make use of the state space and the action space, and we will assign, in the defined order, a value from 0 to 5 to each action. For each state, the optimal action is the one that adds the most to the total reward.
When the environment is defined, a reward table or a matrix [number of states as rows, number of actions as columns] is created, named the Q-table. The table is used by the agent to acquire knowledge from it and hold the values of action-value functions, initially populated with zeros, but, during the training, with values that will optimize the agent strategy for the maximum total reward.
The first step of the algorithm is the creation of Q-table with 0 values. Secondly, the algorithm will iterate trough each state and select any of the actions that are available for the chosen state. As a result of the action taken, the agent „goes" to the next state and sees which action has the greatest Q-value.
The Q-table will be updated afterwards with values obtained from the Boltmann equation (2) and the next state will become the current state. This will be repeated until the goal is reached. After training with a large data set, it is proven that the agent has effectively learned the best move in a predefined matrix. Over time, the hyper parameters as the learning rate and the exploration level should decrease, as the gained knowledge increases, and the discount factor would increase as well because receiving the desired reward very fast is preferable.
Q-Learning learns the optimal policy even when actions are selected according to a more exploratory or even random policy [8].

5 Implementation

5.1 Multi Armed Bandit Algorithm Evaluation

For the Multi-Armed Bandit algorithm, we make use of the *pandas* library in Python. *Pandas* is a very powerful tool that enables a lot of tools for data processing with very high optimization.
Because of the lack of data that is needed for the conceptual problem described, the method of implementing the algorithm relies on a randomly generated a set of data.
For the sake of simplicity, we assume already observed data for the plots, and the yield already computed based on the model

described above[6]. We encode the obtained yield over fixed values as a good one, and we annotate it with „1", and whatever is below the value, with „0".
Using pandas, a DataFrame object is created with the randomly obtained values for 200 observations.

```
data = {}
data['A1'] = [random.randint(0,1) for x in range(200)]
data['A2'] = [random.randint(0,1) for x in range(200)]
data['A3'] = [random.randint(0,1) for x in range(200)]
data['A4'] = [random.randint(0,1) for x in range(200)]
data['A5'] = [random.randint(0,1) for x in range(200)]
data['A6'] = [random.randint(0,1) for x in range(200)]
data['A7'] = [random.randint(0,1) for x in range(200)]
data['A8'] = [random.randint(0,1) for x in range(200)]
data['A9'] = [random.randint(0,1) for x in range(200)]
data['A10'] = [random.randint(0,1) for x in range(200)]
data = pd.DataFrame(data)
print("\n\nPlot Data = ", data)
```

A list is initialized for the rewards associated with each plot, and one for all the penalties that belong to the plots.
For each observation, we iterate through each machine and based on the highest random beta distribution, the plot selected is updated with the plot/machine used. Once the plot is selected, the data corresponding to it is verified and we updated the list of rewards/penalties accordingly.

```
for m in range(0,total_observations):
    plot = 0
    beta_max = 0
    for i in range(0, machines):
        beta_d = random.betavariate(rewards[i]+1, penalties[i]+1)
        if beta_d > beta_max:
            beta_max = beta_d
            plot = i
    machine_used.append(plot)
    reward = data.values[m,plot]
    if reward == 1 :
         rewards[plot] = rewards[plot] +1
    else:
         penalties[plot] = penalties[plot] +1
    total_reward = total_reward + reward

print("\n\nRewards by machine = ", rewards)
print("\n\nTotal rewards = ", total_reward)
print("\n\n Machine used at each round: \n", machine_used)

plt.bar(['A1','A2', 'A3', 'A4', 'A5', 'A6','A7', 'A8', 'A9', 'A10'], rewards)
plt.title('MABP')
plt.xlabel('Bandits')
plt.ylabel('Rewards By Machine')
plt.show()
```

The output is :

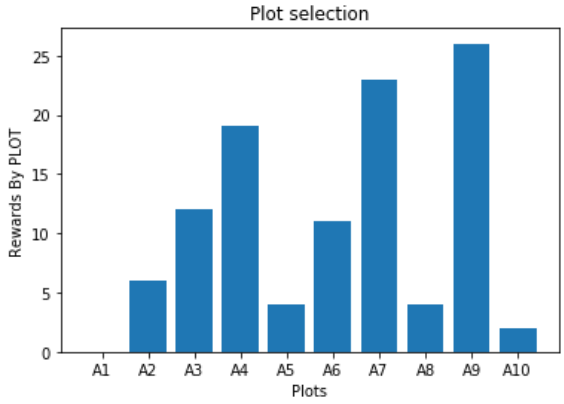

Fig. 1. Plot selection

In order to make sure that the algorithm selected the most optimal plot over time, we make a histogram of the plot that we use over time.

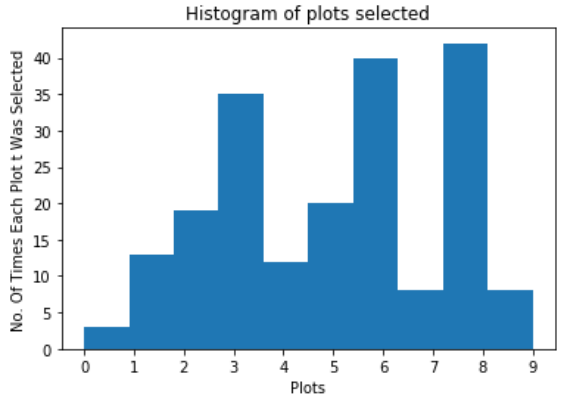

Fig. 2. Histogram of selected plots

By comparing the two graphs, we can see that the optimal plot was selected every time.

5.2 Q-Learning Algorithm evaluation

According to the GitHub repository, openAI Gym is a toolkit for developing and comparing reinforcement learning algorithms. For the implementation, the available collection of environments is useful for testing the agent, because the library also provides the required information as in states, scores or actions. In

openAI Gym, the environment replies with rewards, namely, scores.
We make use of *Env*, the core gym interface, and the predefined methods available: reset, step and render. Fortunately, openAI Gym provides a built-in environment, named „Taxi-v2", but the environment uses only a matrix of 25 possible agent locations and four possible destinations/locations.

In order to extend the locations, a new environment is created in openAI Gym. For achieving this, the environment is registered by calling gym's *register()* function, and by running the command *pip install –e* that takes as argument the location of the setup file where we defined the new environment. The custom made environment will be available with the call of:

```
env = gym.make('truck-v0')
env.render()
```

The focus is to break down the agent's learning experience into episodes.
Each episode starts by setting the first state of the agent randomly selected from the distribution. The agent iterates through episodes with the scope of maximizing the expectation of total reward/episode.

For our TruckEnv, we initilize „the field", like this matrix:

```
FIELD = [
    ['-','-','-','-','-','-','-','-','-','-','-','-','-','-','-'],
    ['-','1','0','1','0','0','1','0','0','0','0','0','0','-'],
    ['-','0','0','0','0','0','0','0','0','0','0','0','0','-'],
    ['-','0','0','0','0','0','0','0','0','0','0','0','0','-'],
    ['-','0','0','0','0','0','0','0','0','0','0','0','0','-'],
    ['-','1','0','0','0','0','1','0','0','0','0','0','0','-'],
    ['-','-','-','-','-','-','-','-','-','-','-','-','-','-','-']
```

Fig. 3. Field matrix

The state space is represented by :"truck_row","truck_col", „crop_location", „silo_destination".

For our example, we use a matrix with 5 possible crop locations and silo destinations. So, the total number of states is 6 * 6 [the possible positions of the truck in the matrix] * 5[the possible crop locations] * 5 [the silo destinations] = 900 possible states;
When the environment interface is built, the initial matrix of states and actions is created.
In order to view the structure, with the call of *env.P()*, we can see for each of the five possible actions the probability, the next state in which the agent will be if the action at the indicated index is taken, the amount of the reward[based on the type of the action and the position], and a boolean value, namely, „done", which indicates if the episode is successful.
In the environement, the P structure is initialize like this:

P = {state: {action: []
for action in range(number_of_actions)}
for state in range(number_of_states)}

We leave the other functions of the environment as they are defined in openAI gym default environment, because, basically, the truck will use the same context to train.

For training, the following hyperparameters were used:

- ***alpha***, the learning rate as 0.1
- ***gamma***, the discount factor and we set like this the importance of the future reward as 0.5
- ***epsilon***, the quantification of the exploration phase

Explaining it in a simple manner, it would be the decision of whether to check random actions or to make use of the already computed values in the Q-table. Epsilon is set in the code as 0.1

The number of episodes is set to 100000 for the training. We iterate through all the

episodes, and we reset the environment at each iteration to get a clean step.
For each step, we check the epsilon value against a random value from 0 and 1 and decide if the action is a random one or we just exploit the already known actions that have the greatest value in the Q-table.

Afterwards, the action is taken and the next step becomes the current one, the reward and the „done" boolean value are actualized along with the Qtable value for the specific state and action based on the formula described above.

```
for episode in range(total_episodes):
    state = env.reset()
    step = 0
    done = False
    for step in range(steps):
        if random.uniform(0, 1) < epsilon:
            action = env.action_space.sample() # Explore action space
        else:
            action = np.argmax(qtable[state,:]) # Exploit learned values
        # Take the action and observe the outcome state and reward
        new_state, reward, done, info = env.step(action)
        # Q(old):= Q(old) + lr [R + gamma * max Q(new) - Q(old)]
        qtable[state, action] = qtable[state, action] + alpha * (reward + gamma *
                                np.max(qtable[new_state, :]) - qtable[state, action])
        # Let new_state be state
        state = new_state
        # If done : finish episode
        if done == True:
            break
    # Reduce epsilon (because we need less and less exploration)
    epsilon = min_epsilon + (max_epsilon - min_epsilon)*np.exp(-decay_rate*episode)

if episode % 100 == 0:
    clear_output(wait=True)
```

For evaluating the performance of the agent, we make use of three indicators: the average number of steps taken to reach the destination, the average number of rewards and penalties per move, and so, after the training, we iterate through the range of episodes until the done indicator is true. By selecting the action only with the use of the Q-table, we can calculate these parameters and observe the performance.

```
env.reset()
rewards = []

total_rewards= 0
total_penalties = 0
total_test_episodes = 50
total_steps = 0
for i in range(total_test_episodes):
    state = env.reset()
    rewards, penalties = 0, 0
    done = False
    while not done:
        action = np.argmax(qtable[state])
        state, reward, done, info = env.step(action)
        if reward == -10:
            penalties += 1
        else:
            rewards += 1
        steps += 1

    total_penalties += penalties
    total_steps += steps
    total_rewards += rewards

print(f"Results after {total_test_episodes} episodes:")
print(f"Average timesteps per episode: {total_steps / total_test_episodes}")
print(f"Average penalties per episode: {total_penalties / total_test_episodes}")
print(f"Average rewards per episode: {rewards / total_test_episodes}")
```

The output is as follows:

```
Average timesteps per episode: 405.18
Average penalties per episode: 0.0
Average rewards per episode: 0.34
```

Fig. 4. Performance of the agent

6 Conclusions

The evolution of Big Data and Machine Learning will change the methods of farm management and is actually changing the research methods. Optimization is definitely improved by the prevalence of data and rapid estimation of causality relationship of inputs is overpassing the traditional approaches. Also, the adoption of data-driven technologies will play a big role in conserving resources and expanding the returns. Analysis of data from software that manages irrigation reduces water consumption and impacts environmental management. Predictions based on the historical data are being replaced with a comprehensive analysis of the crops, based

on real-time input. Machines can also classify and detect plant disease reducing costs this way and improving the quality of the crops. Progress in machine learning has been driven by low-cost computation opportunities as well by the availability of online resources, data and the development of learning algorithms.

References

[1] D. J. Russo, B. Van Roy, Benjamin, A. Kazerouni, A. Osband, "A Tutorial on Thompson Sampling. Foundations and Trends", *Machine Learning*, 11. 10.1561/2200000070, 2017

[2] S. Sukhbaatar, A. Szlam, R. Fergus. "Learning Multiagent Communication with Backpropagation", *NIPS*, 2016.

[3] P. Sterling, S. Laughlin, "*Principles of Neural Design*", MIT Press, Cambridge, MA, 2015

[4] CC Bennett, K Hauser, "*Artificial intelligence framework for simulating clinical decision-making: A Markov decision process approach*", Artificial intelligence in medicine, Elsevier 2013

[5] D. J. Russo, B. Van Roy, A. Kazerouni, I. Osband, Z. Wen (2018), "*A Tutorial on Thompson Sampling*", Foundations and Trends® in Machine Learning: Vol. 11: No. 1, pp 1-96

[6] A.P. Gaspar, P.D. Mitchell, S.P. Conley, "Economic risk and profitability of soybean fungicide and insecticide seed treatments at reduced seeding rates". *Crop Sci*, 55, 924-933, 2015

[7] B.Gutiérrez, L. Peter, T. Klein, C. Wachinger, "A Multi-Armed Bandit to Smartly Select a Training Set from Big Medical Data", *Computer Vision and Pattern Recognition*, arXiv:1705.08111, 2017

[8] R. S. Sutton, A.G. Barto, "*Reinforcement Learning: An Introduction*", MIT Press, 1998

Gabriela ENE has graduated the Faculty of Cybernetics, Statistics and Economic Informatics in 2015. Currently she is a PhD Student at the same university. Main fields of interest are big data technologies, artificial intelligence, web development, algorithms and data structures.