Poster Title: Village Game using Reinforcement Learning

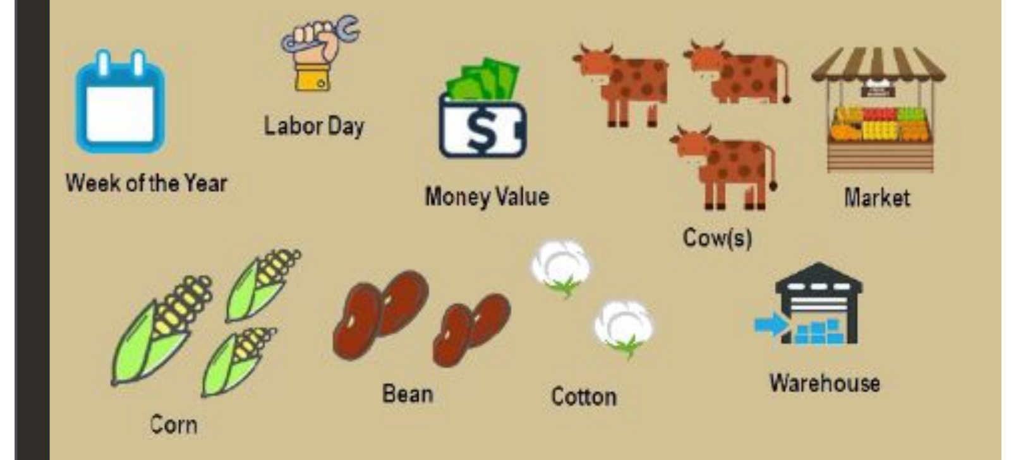
Megala Anandakumar, Rakesh Talanki, Bohdan Metchko Junior

nmegala9@stanford.edu, rakeshTL@stanford.edu, bohdanjr@stanford.edu
Stanford University

Stanford
Artificial Intelligence

Abstract

Problem: Convert board game called Village Game into an AI Problem
Objective: Build an Agent using RI models to maximize the earnings by finding an optimal policy and come close to Oracle Solution.



Motivation

Motivation 1: We think providing practical education in a game environment can promotes social equality and can have a profound effect especially when done at a young age. This game when developed completely can teach the children the responsibilities of a family and how various decisions can affect their livelihood

Motivation 2: We want to pick a problem in Reinforcement Learning space using Q-Learning technique applying what is taught in the course.

Presentation: YouTube

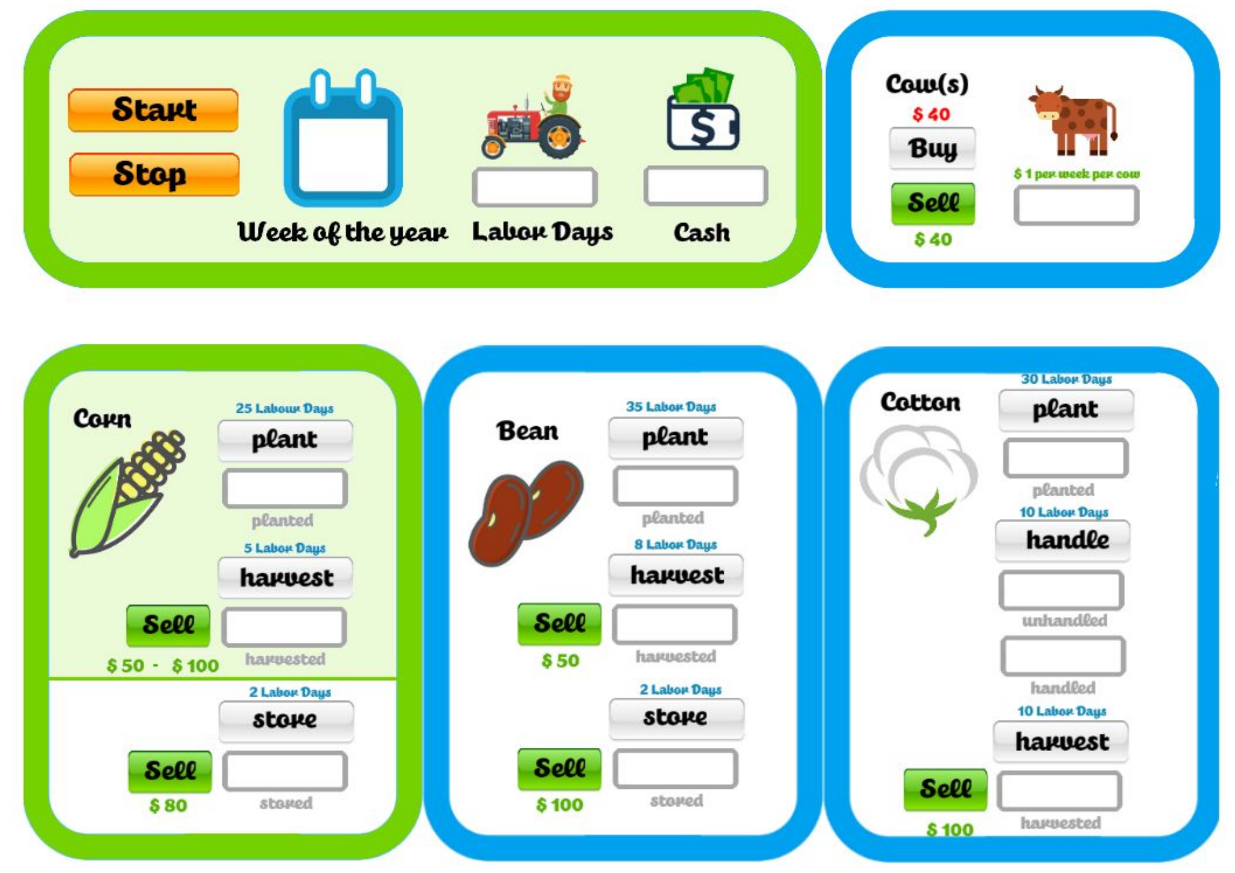
Sourcecode: Github

Stanford University

Features

States
- Week
- Planted Corn
- Harvested Corn
- Money (Poor, Medium, Rich)
- Labor Days (Low, Medium, High)

Actions
- Idle
- Plant Corn
- Harvest Corn
- Sell Corn



The original game (blue and green) and the simplified one (green)

Input and Output

- Agent plays in an unknown environment.
- Actions can happen stochastically.
- Rewards are time delayed.
- Output is money earned over many iterations of the game.
- States (Full game):

Key	*Description*	*Values*	*Values (Simplified)*
Week	The current week of the year	0-104	0-13
Planted Corn	Units of planted corn	0-3	0-3
Planted Cotton	Units of planted cotton	0-3	
Harvested Corn	Units of harvested corn	0-3	0-3
Harvested Cotton	Units of harvested cotton	0-3	
Unhandled Cotton	Unhandled units of Cotton	0-3	
Handled Cotton	Handled units of Cotton	0-3	
Stored Corn	Stored units of Corn	0-3	
Cow	Units of cow	0-3	
Money Value	It is an integer between zero and ten.	0-9	1,2,3
Labor Days Value	It is an integer between zero and ten	0-9	1,2,3

Solution 1: Q Learning (Q Table)

- Q_learning is model-free algorithm that will help us to find the most optimum policy.

Algorithm: Q-learning [Watkins/Dayan, 1992]

On each (s, a, r, s'):

$$\hat{Q}_{\text{opt}}(s,a) \leftarrow (1-\eta)\underbrace{\hat{Q}_{\text{opt}}(s,a)}_{\text{prediction}} + \eta\underbrace{(r + \gamma\hat{V}_{\text{opt}}(s'))}_{\text{target}}$$

Recall: $\hat{V}_{\text{opt}}(s') = \max_{a' \in \text{Actions}(s')} \hat{Q}_{\text{opt}}(s', a')$

- **Our algorithm took 10,000 episodes to learn, with: η = 0.01 and γ = 0.95. Reward earned using this model averaged to \$178 compared to Oracle solution of \$225.**
- We first explored in the beginning and then did exploitation towards the end using a dynamic value of epsilon.

Algorithm: epsilon-greedy policy

$$\pi_{\text{act}}(s) = \begin{cases} \arg\max_{a \in \text{Actions}} \hat{Q}_{\text{opt}}(s,a) & \text{probability } 1-\epsilon, \\ \text{random from Actions}(s) & \text{probability } \epsilon. \end{cases}$$

Solution 2: Function Approximation

- Unlike Q-Learning, Function approximation deals with generalization, one of the most important aspects of learning.

Algorithm: Q-learning with function approximation

On each (s, a, r, s'):

$$\mathbf{w} \leftarrow \mathbf{w} - \eta[\underbrace{\hat{Q}_{\text{opt}}(s,a;\mathbf{w})}_{\text{prediction}} - \underbrace{(r + \gamma\hat{V}_{\text{opt}}(s'))}_{\text{target}}]\phi(s,a)$$

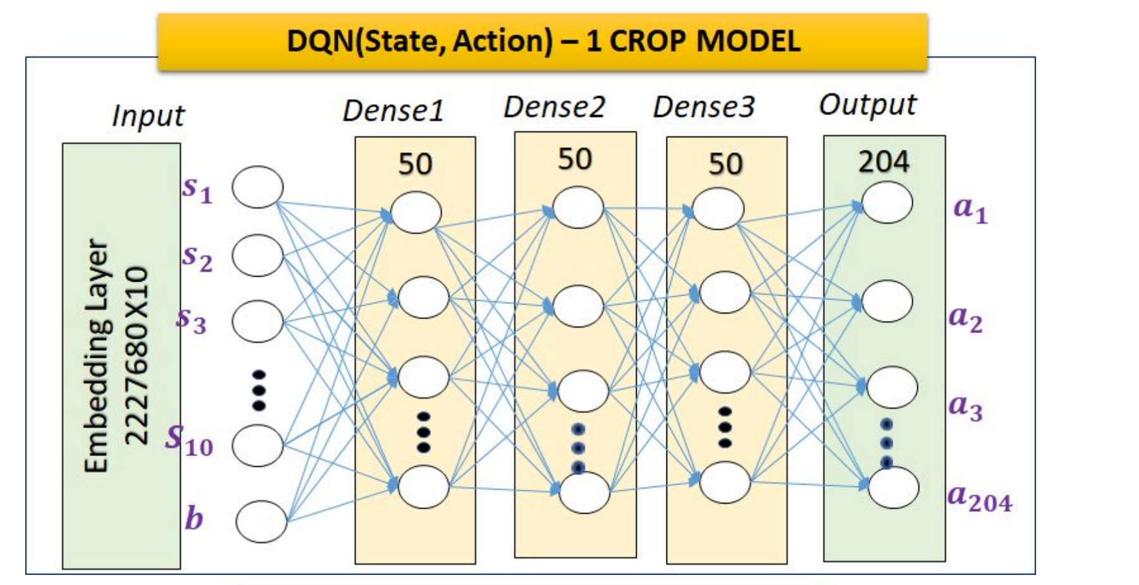
- In our solution, we experimented with 2 sets of features. Reward earned using a collection of state-action (s,a) pairs averaged to \$150.
- Using customized features the reward earned averaged to \$ 176.

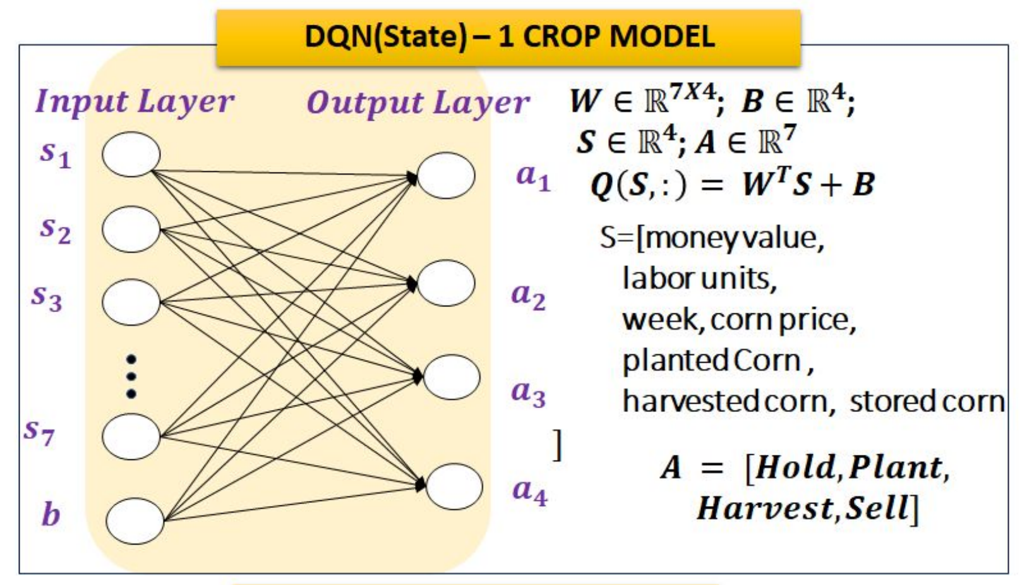
Feature vector $\phi(x) \in \mathbb{R}^d$

action, money	: 1.0
action, plantedcorn	: 1.0
action, plantedcorn, cornprice	: 1.0
action, harvestedcorn	: 1.0
action, harvestedcorn, cornprice	: 1.0
action, harvestedcorn, money	: 1.0

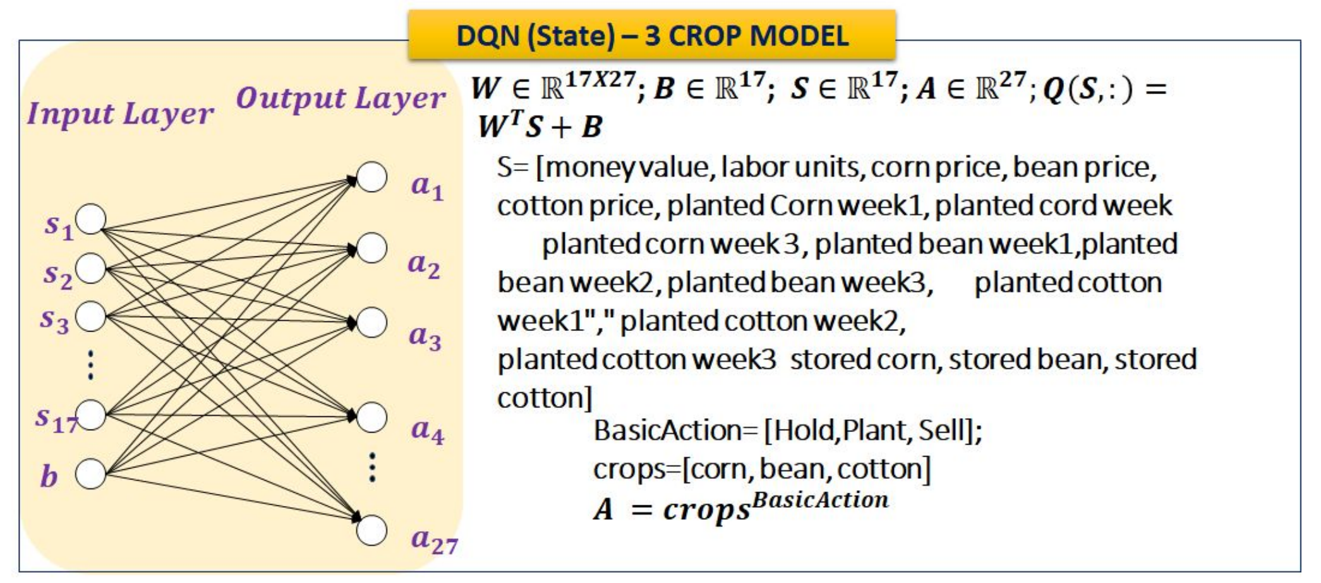
Solution 3: Deep Q Network

We used 3 neural network models to get Q-value function. i) DQN States, Actions as inputs for 1 crop ii) DQN states for 1 crop iii)DQN states for 3 crops



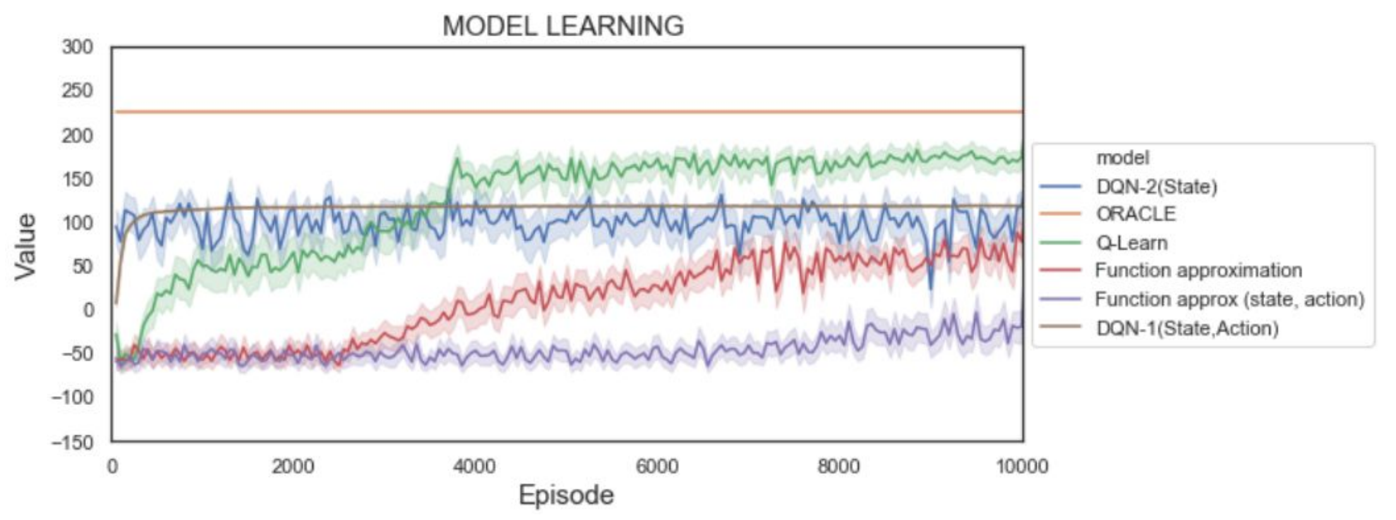


$W \in \mathbb{R}^{7X4}$; $B \in \mathbb{R}^4$;
$S \in \mathbb{R}^4$; $A \in \mathbb{R}^7$
$Q(S,:) = W^T S + B$
S=[money value, labor units, week, corn price, planted Corn, harvested corn, stored corn]
$A = [Hold, Plant, Harvest, Sell]$

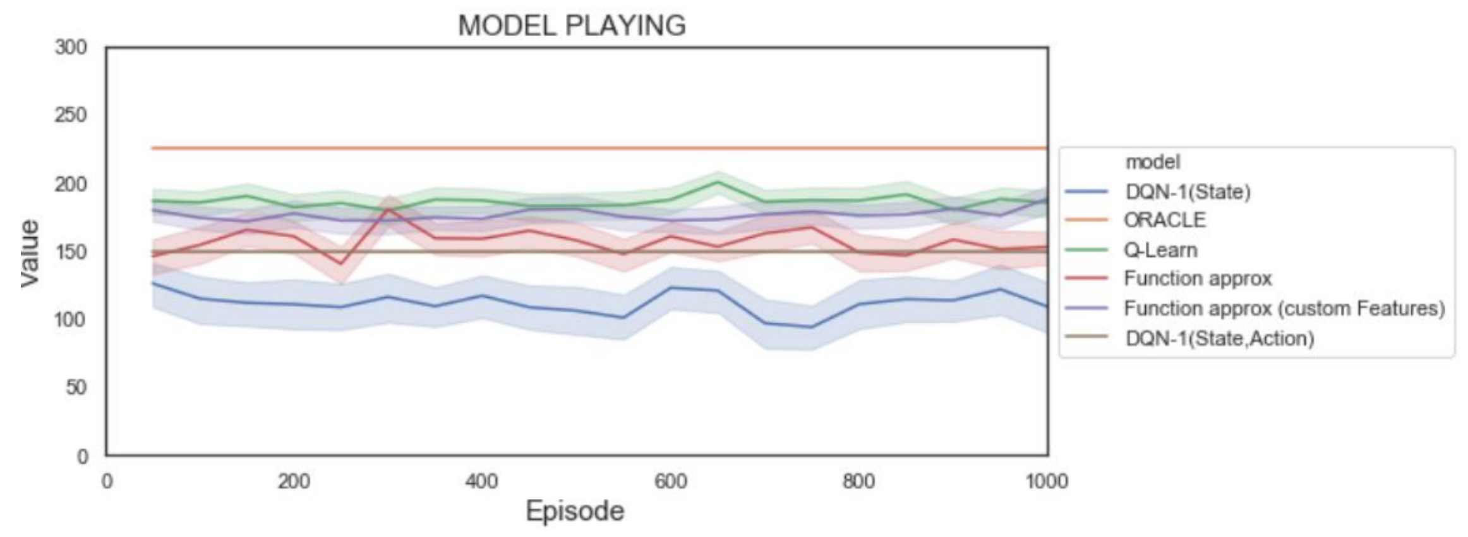


$W \in \mathbb{R}^{17X27}$; $B \in \mathbb{R}^{17}$; $S \in \mathbb{R}^{17}$; $A \in \mathbb{R}^{27}$; $Q(S,:) = W^T S + B$
S= [money value, labor units, corn price, bean price, cotton price, planted Corn week1, planted cord week planted corn week3, planted bean week1, planted bean week2, planted bean week3, planted cotton week1"," planted cotton week2, planted cotton week3 stored corn, stored bean, stored cotton]
BasicAction= [Hold, Plant, Sell];
crops=[corn, bean, cotton]
$A = crops^{BasicAction}$

Solution Comparison



Q Learning provides the most optimal solution as noticed in the Learning and Playing graphs.



Results and Discussion

In our implementation we have used Python3, OpenAI Gym, Jupyter Notebook and Tensorflow.

Q-Learning: Performed best (\$184) compared to Oracle solution (\$225) for a small number of states and actions. Disadvantage is once we add more crops and actions, this model is not scalable.

Function Approximation:
i) State, Action - Good performance (\$151), but training. Needs improvement in definition of features to get an optimal solution

ii) Custom Features - Better performance (\$174), and training stable because of well defined features. Tuning features is manual work and can be tedious.

DQN
i) State Action based - Good performance (\$152), but number of states is too big (Q Learning). Hence training is slow. With growing number of states, this solution can become very unusable due to slowness.

ii) State based 1 crop model - Poor performance (\$112) due to not well developed features in function approximation. Improving input features and reward function will give better results

iii) State based 3 crop model - Good performance even when solution is increased to 3 crops.

Future work:
When the full fledged game is built, we recommend to use DQN with function approximation as this gives good performance and can manage States better than other Models we have used here.

References and Recognition

- We would like to acknowledge our mentor Zach Barnes for his guidance in implementing our solutions.
- Deep Reinforcement Learning for Financial Trading Using Price Trailing, Zarkias et all, IEEE, ICASSP 2019
- Reinforcement Learning: Deep Q-Network (DQN) with Open AI Taxi
- OpenAI Gym Style Tic-Tac-Toe Environment