Manage a farm with Reinforcement Learning

Elisa Stabilini

October 31, 2024

Università degli Studi di Milano - Department of Physics

Table of contents

Environment definition

Problem definition

- Initial budget: 2000 €
- Cost of growing wheat: 20 €
- Wool selling profit: 10 €
- Wheat selling profit: 50 €
- Storm probability: 30%
- Sheep population with binomial distribution

$$N_t = N_0(1.15)^t$$

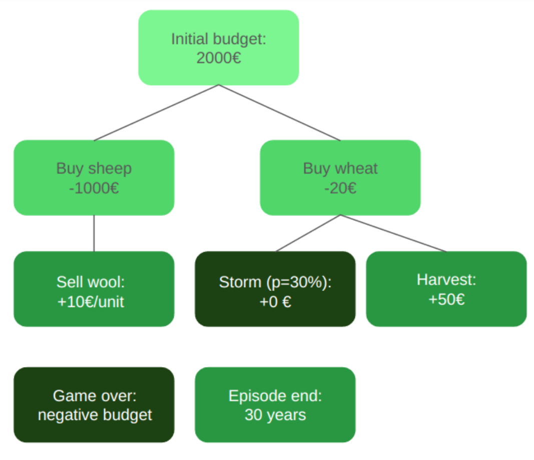

Figure 1: Problem model

Environment key features

- observation space: `budget`, `sheep`, `year`
- Reward definition `r = current_budget - initial_budget`
- Normalized reward: `norm_reward = (reward - min_reward) / (max_reward - min_reward)` where
  - `min_reward = -self.initial_budget` describes the worst case, losing all money
  - `max_reward = self.max_budget - self.initial_budget` describes the best case, reaching the max budget
- State normalization to improve learning stability [4]:
  - `norm_budget = budget/max_budget` with `max_budget = 200000`
  - `norm_sheep = sheep_count/max_sheep` with `max_sheep = 70`

Agents definition

Deep Q-Learning algorithm

- Value function approximation (VFA):

$$\hat{Q}(s, a, \mathbf{w}) \rightarrow \mathbf{x}(s, a)^T \mathbf{w}$$

- Update rule for Q-learning:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right]$$

- Deep Q-Learning: uses deep neural network instead of linear approximation for $\hat{Q}$
- DQN update rule:

$$\Delta(\mathbf{w}) = \alpha \left[r_t + \gamma \max_{a_{t+1}} \hat{Q}(s_{t+1}, a_{t+1}, \mathbf{w}) - \hat{Q}(s_t, a_t, \mathbf{w}) \right] \nabla_{\mathbf{w}} \hat{Q}(s_t, a_t \mathbf{w})$$

Deep Q-Learning agent

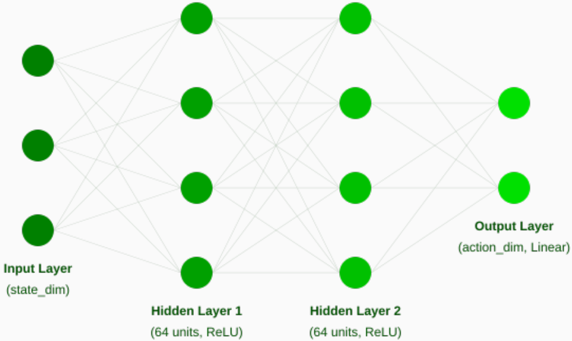

Figure 2: Training and target network architecture

- training & target network
- ϵ-greedy strategy
- ϵ-decay over time
- experience replay

REINFORCE algorithm

Policy gradient methods:

- Parameterized policy $\pi_\theta(s, a) = \mathbb{P}(a|s; \theta)$
- Parameters update: $\theta_{t+1} = \theta_t + \alpha \nabla \mathcal{J}(\theta_t)$ (gradient ascent)
- Goal: update of a parameterized policy to maximize the expected total return

$$\nabla_\theta \mathbb{E}_{\pi_\theta} G(\tau) = \nabla_{\theta_\theta} \sum_\tau P(\tau|\theta) G(\tau)$$

- REINFORCE: Monte-Carlo Policy Gradient Control with update step

$$\theta \leftarrow \theta + \alpha \gamma^t G \nabla_\theta \log(\pi(a_t|a_t; \theta))$$

REINFORCE agent

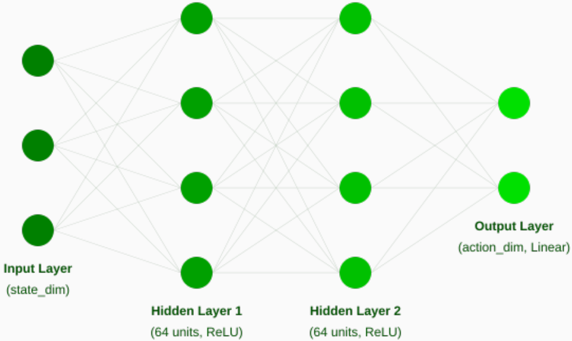

Figure 3: Policy network architecture

- Policy network outputs a probability distribution over all possible actions for a given state
- Action randomly chosen according to probability distribution

Results

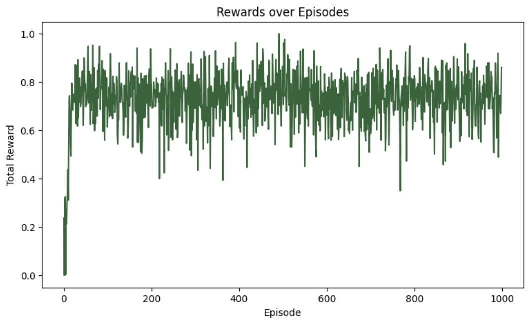

Figure 4: Normalized rewards across episodes

Figure 5: Comparison between raw rewards across different episodes

REINFORCE training

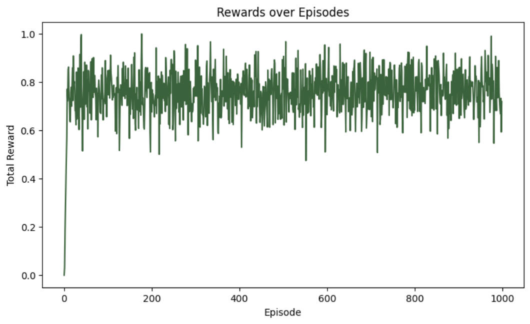

Figure 6: Normalized rewards across episodes

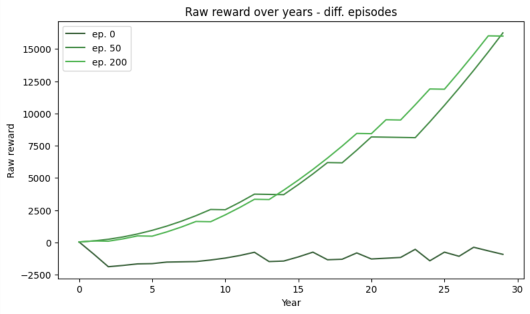

Figure 7: Comparison between raw rewards across different episodes

REINFORCE vs DQN training

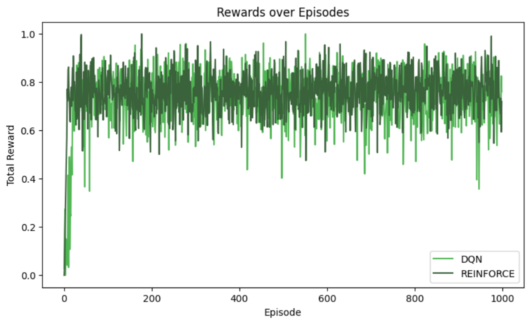

Figure 8: Normalized rewards across episodes comparison

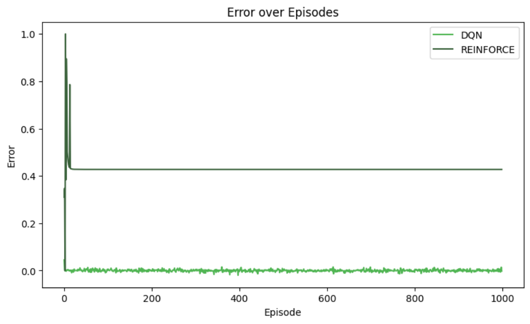

Figure 9: Average errors across episodes comparison

Errors evaluation REINFORCE vs DQN

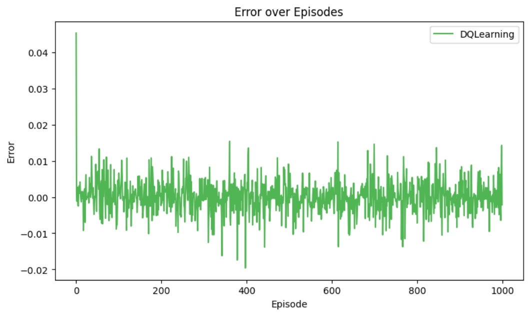

Figure 10: Average errors for DQN training

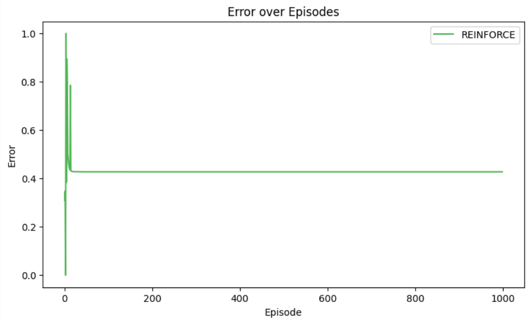

Figure 11: Average errors for REINFORCE training

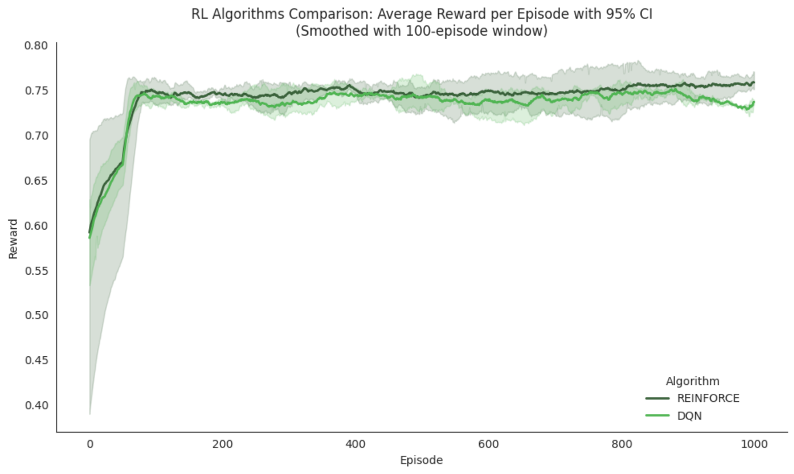

Figure 12: Algorithm comparison over different trainings

Further improvements

Possible improvements & developments - I

Learning strategy:

- Use REINFORCE with advantage
- Use REINFORCE with baseline
- Test actor-critic methods
- Test dueling architecture
- Test Prioritized experience replay

Possible improvements & developments - II

Environment setting:

- Test pseudo-tabular method (e.g. setting reward between -1 and 1 with bins related to the profit percentage)
- Adaptive reward normalization [7]
- More training (to study convergence)

Algorithmic efficiency:

- Hyperparameters optimization using `optuna`
- Improving performance with parallelization

Learning evaluation:

- Test different metrics for more objective learning evaluation

Thanks for your attention

Backup slides

Training example without experience replay

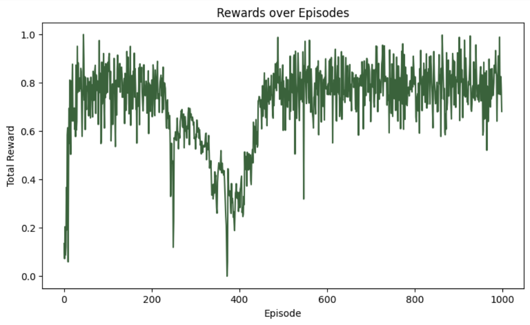

Figure 13: Rewards in DQN training without experience replay

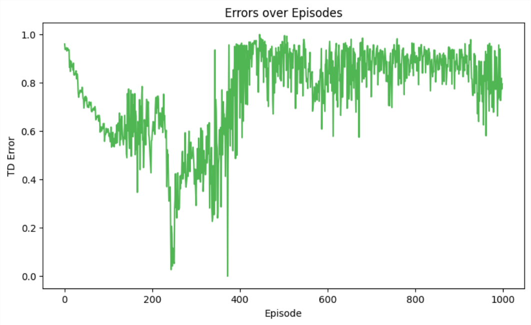

Figure 14: TD error in DQN training without experience replay

Tabular Q Learning example

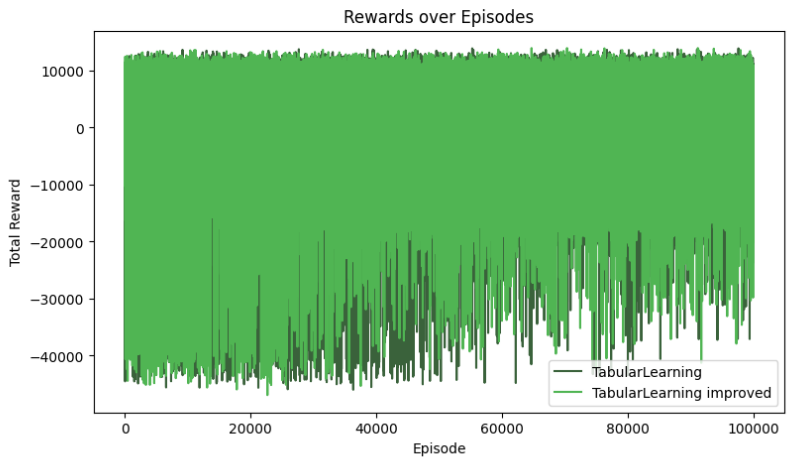

Figure 15: Rewards during training with tabular QLearning

Agent initialization examples

- Deep Q Learning agent:

```
def __init__(self, env, model=None, learning_rate=0.001,
gamma=0.99, epsilon=1.0, epsilon_decay=0.995, epsilon_min=0.01,
batch_size=32, memory_size=10000):
```

- REINFORCE agent:

```
def __init__(self, env, learning_rate=0.01, gamma=0.99)
```

Data organization - I

```
TrainedDQLearning/
├── Data/
│   ├── env_history_<timestamp>.npz
│   ├── episode_info_<timestamp>.pkl
│   ├── episode_length_<timestamp>.npy
│   ├── rewards_<timestamp>.npy
│   └── training_errors_<timestamp>.npz
│
```

Data organization - II

```
├── Model/
│   ├── DQN_final_<timestamp>/
│   │   ├── assets/
│   │   ├── fingerprint.pb
│   │   ├── saved_model.pb
│   │   └── variables/
│   └── DQN_final_<timestamp>.keras
│
```

Error calculation

Error calculation for DQN algorithm:

- Single step error:

$$\Delta_t = (r_t + \gamma \max_{a'} Q_{target}(s_{t+1}, a')) - Q(s_t, a_t)$$

- Episode error: $Err_{episode} = \frac{1}{N} \sum_{t=1}^{N} \Delta_t$

Error calculation for REINFORCE algorithm:

$$L(\theta) = -\sum_{t=0}^{T} G_t \cdot \log \pi_\theta(a_t|s_t)$$

François Chollet et al.
Keras.
`https://github.com/fchollet/keras`, 2015.

Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant.
Array programming with NumPy.
Nature, 585(7825):357–362, September 2020.

Packages and libraries references ii

J. D. Hunter.
Matplotlib: A 2d graphics environment.
Computing in Science & Engineering, 9(3):90–95, 2007.

Clare Lyle, Zeyu Zheng, Khimya Khetarpal, James Martens, Hado van Hasselt, Razvan Pascanu, and Will Dabney.
Normalization and effective learning rates in reinforcement learning, 2024.

Richard S Sutton and Andrew G Barto.
***Reinforcement Learning*.**
Adaptive Computation and Machine Learning series. Bradford Books, Cambridge, MA, 2 edition, November 2018.

Packages and libraries references iii

- Mark Towers, Jordan K. Terry, Ariel Kwiatkowski, John U. Balis, Gianluca de Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Arjun KG, Markus Krimmel, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Andrew Tan Jin Shen, and Omar G. Younis.
  Gymnasium, March 2023.
- Hado van Hasselt, Arthur Guez, Matteo Hessel, Volodymyr Mnih, and David Silver.
  Learning values across many orders of magnitude, 2016.
- Michael L. Waskom.
  seaborn: statistical data visualization.
  Journal of Open Source Software, 6(60):3021, 2021.