

NAIL139 AI for Computer Games

Lecture 10: Deep Reinforcement Learning 1

Adam Dingle

December 5, 2025

Textbooks

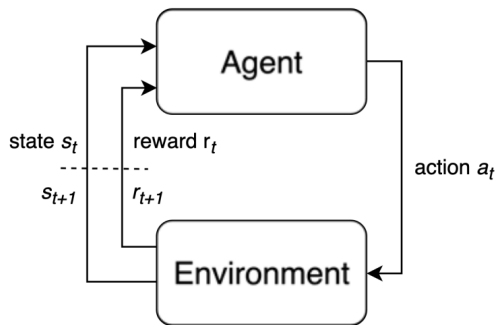
- ▶ Foundations of Deep Reinforcement Learning (Graesser/Keng, 2020)
- ▶ Reinforcement Learning: An Introduction (Sutton/Barton, 2018)
- ▶ Deep Learning: A Visual Approach (Glassner, 2021)
- ▶ Understanding Deep Learning (Prince, 2023)

Reinforcement learning

- ▶ Learning by interacting with the environment
- ▶ Maximizing a numerical reward signal
- ▶ Applicable to many problems
 - ▶ playing a game, driving a car, controlling a robot...
- ▶ Different from supervised, unsupervised learning

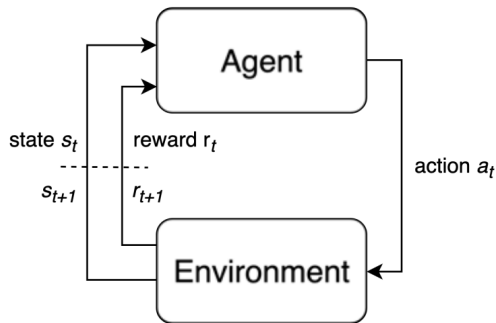
Reinforcement learning: model

- ▶ An **agent** is in an **environment**
- ▶ The environment has a current **state**
- ▶ At each **time step** t the agent observes state s_t , chooses an **action** a_t
- ▶ The environment produces the next state plus a **reward** r_t



Reinforcement learning: model

- ▶ A **policy** is a function from a state to an action
- ▶ **Objective**: maximize the (weighted) sum of rewards
- ▶ At each time step (s_t, a_t, r_t) is an **experience**
- ▶ An **episode** lasts from time $t = 0$ until the environment **terminates**
- ▶ A **trajectory** is a sequence of experiences over an episode



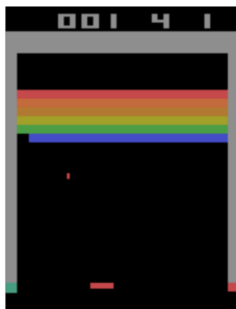
A simple environment: CartPole (OpenAI Gymnasium)

- ▶ Objective: keep the pole upright for 200 steps
- ▶ State: [cart position, cart velocity, pole angle, pole angular velocity] (e.g. $[-0.037, 0.032, -0.036, 0.038]$)
- ▶ Action: 0 to move left, 1 to move right
- ▶ Reward: +1 for every time step the pole stays upright
- ▶ Termination: when pole falls more than 12 degrees or at time $t = 200$



A more complex environment: Atari Breakout

- ▶ Objective: maximize game score
- ▶ State: digital image, 160 x 210 pixels
- ▶ Action: 0 = no action, 1 = launch ball, 2 = move right, 3 = move left
- ▶ Reward: score difference between consecutive states
- ▶ Termination: when all game lives are lost



States, actions, rewards

- ▶ $s_t \in S$ is the **state** ($S =$ state space)
- ▶ $a_t \in A$ is the **action** ($A =$ action space)
- ▶ $r_t = R(s_t, a_t, s_{t+1})$ is the **reward** ($R =$ reward function)
- ▶ State space and action space may be **discrete** or **continuous**

Markov property

- ▶ **Markov property**: assume that next state s_{t+1} depends only on previous state s_t and action a_t
- ▶ $s_{t+1} \sim P(s_{t+1} | s_t, a_t)$
 - ▶ \sim means “is sampled from a probability distribution”

Markov decision processes

- ▶ An MDP (**Markov decision process**) is defined by
 - ▶ S : set of states
 - ▶ A : set of actions
 - ▶ $P(s_{t+1} | s_t, a_t)$: state transition function
 - ▶ $R(s_t, a_t, s_{t+1})$: reward function
- ▶ Some environments are only partially observable
 - ▶ POMDP = partially observable Markov decision process
 - ▶ Agent receives only a subset of the state information

The agent's objective

- ▶ The **return** after time t may be the sum of future rewards:

$$R_t = r_t + r_{t+1} + r_{t+2} + \dots + r_T$$

- ▶ But we will often apply a discount factor $\gamma \in [0, 1]$:

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots + \dots + \gamma^{T-t} r_T$$

- ▶ If $\gamma = 0$, we only care about the next reward
- ▶ If $\gamma = 1$, present and future rewards are the same
 - ▶ Can't be used with an infinite time horizon!
- ▶ The agent wants to maximize the **expected return**

$$J_t = \mathbb{E}_{\mathcal{T} \sim \pi} [R_t]$$

Reinforcement learning vs. supervised learning

- ▶ No oracle
 - ▶ We don't have a “correct” answer for every data point
 - ▶ We take an action and receive a reward, but don't know whether another action was better
 - ▶ We only receive rewards about states we experienced
- ▶ Sparse feedback
 - ▶ Reward is often 0
 - ▶ **Credit assignment problem**: which past action ultimately caused a reward to happen?
 - ▶ So less sample-efficient than supervised learning
- ▶ Data generation
 - ▶ Agent interacts with environment dynamically

Learnable functions

What can the agent learn?

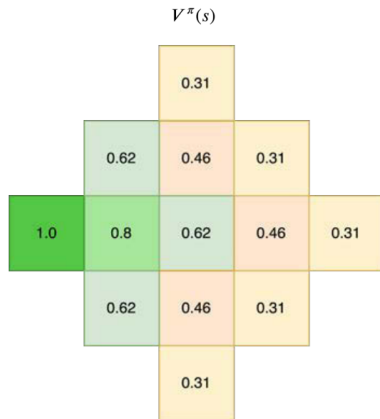
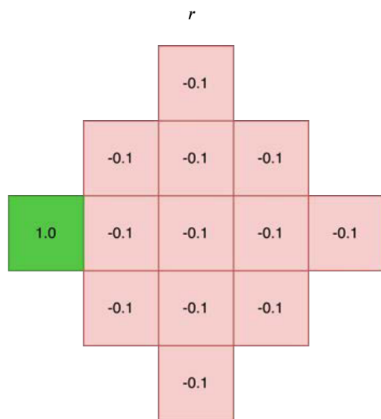
1. A **policy** π that maps states to actions: $a \sim \pi(s)$
2. A **value function** $V^\pi(s)$ or $Q^\pi(s, a)$ that estimates $\mathbb{E}_{\tau \sim \pi}[R_t]$
3. A **model** of the environment: $P(s' | s, a)$

A policy may be stochastic!

Value functions come in two forms:

- ▶ $V^\pi(s)$ estimates the return from being in a state
 - ▶ Not too useful without a model!
- ▶ $Q^\pi(s, a)$ estimates the return from taking an action in a state

Rewards and values in a simple grid world



Some deep reinforcement learning algorithms

- ▶ policy-based: REINFORCE
- ▶ value-based: SARSA, DQN
- ▶ model-based: (MCTS)
- ▶ combined (policy + value, actor-critic): PPO
- ▶ combined (model + policy + value): AlphaZero

Policy-based algorithms (e.g. REINFORCE)

- ▶ advantages
 - ▶ work with either discrete or continuous action spaces
 - ▶ directly optimize the thing we care about
 - ▶ theoretically guaranteed to converge to locally optimal policy
- ▶ disadvantages
 - ▶ high variance
 - ▶ sample-inefficient

Value-based algorithms (e.g. SARSA, DQN)

- ▶ learn either $V^\pi(s)$ or $Q^\pi(s, a)$ (usually $Q^\pi(s, a)$)
- ▶ advantages
 - ▶ more sample-efficient than policy-based algorithms
 - ▶ lower variance
- ▶ disadvantages
 - ▶ no convergence guarantee
 - ▶ usually limited to discrete action spaces

Model-based algorithms

- ▶ Can be given a model of the environment (MCTS, AlphaGo)
- ▶ Or can learn it, e.g. MPC = Model Predictive Control
- ▶ In either case, can search ahead by predicting future trajectory

Combined methods (e.g. PPO, AlphaZero)

- ▶ actor-critic: policy acts, value function criticizes the action
- ▶ AlphaZero: uses MCTS, learns a policy π plus a value function V

On-policy and off-policy algorithms

On-policy: we improve the policy we are using to make decisions

- ▶ like learning from your own experiences
- ▶ advantage: simple
- ▶ disadvantages: sample-inefficient, must discard all data after training
- ▶ examples: REINFORCE, SARSA, PPO, AlphaZero

Off-policy: we can use one policy for decisions while learning a different policy

- ▶ like learning from observing other people's experiences
- ▶ advantage: more sample-efficient
- ▶ disadvantage: may result in divergence via “deadly triad” (function approximation + bootstrapping + off-policy learning)
- ▶ Example: DQN

Deep neural networks: overview

- ▶ Deep neural networks learn **functions** that map inputs to outputs
- ▶ The network maps an input to an output in a forward pass via a series of **layers**
- ▶ The **parameters** (weights) θ of a network determine the function it computes
- ▶ A **loss function** measures the error between actual target and prediction
- ▶ Goal: change parameters during **training** to minimize the loss
- ▶ **Gradient descent** changes the parameters in direction of steepest descent on loss surface

Deep neural networks: training loop for supervised learning

- ▶ Sample a random batch (x, y) from dataset
 - ▶ x = input variable, y = output variable
- ▶ Compute a forward pass with the network: $\hat{y} = f(x; \theta)$
- ▶ Compute the loss $L(\hat{y}, y)$
- ▶ Calculate the gradient (partial derivative) of the loss $\nabla_{\theta}L$ with respect to network parameters
- ▶ Use an **optimizer** to update network parameters along the gradient
 - ▶ stochastic gradient descent: $\theta \leftarrow \theta - \alpha \nabla_{\theta}L$
 - ▶ α is the **learning rate**
 - ▶ more sophisticated optimizers (e.g. Adam) exist

Deep neural networks: software frameworks

- ▶ Frameworks
 - ▶ PyTorch (Meta)
 - ▶ TensorFlow / Keras (Google)
- ▶ Usable from Python and other languages
- ▶ Features
 - ▶ Automatic parallelization on 1 or more GPUs
 - ▶ Automatic differentiation
 - ▶ Libraries of typical network elements (e.g. convolutional units)