Reinforcement Learning

Prof. Gheith Abandah

Reference: *Hands-On Machine Learning with Scikit-Learn, Keras and TensorFlow* by Aurélien Géron (O'Reilly). 2019, 978-1-492-03264-9.

Outline

1. Introduction
2. Policy Search
3. OpenAI Gym
4. Neural Network Policies
5. The Credit Assignment Problem
6. Q-Learning
7. Exercises

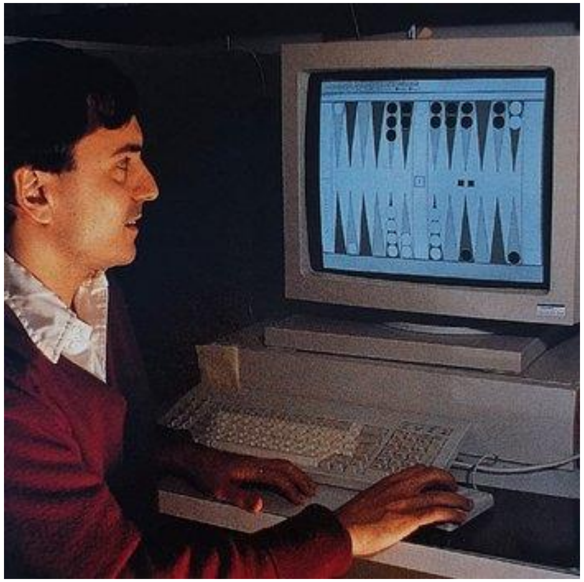
Introduction

- YouTube Video: *An introduction to Reinforcement Learning* from Arxiv Insights

https://youtu.be/JgvyzIkgxF0

1. Introduction – History

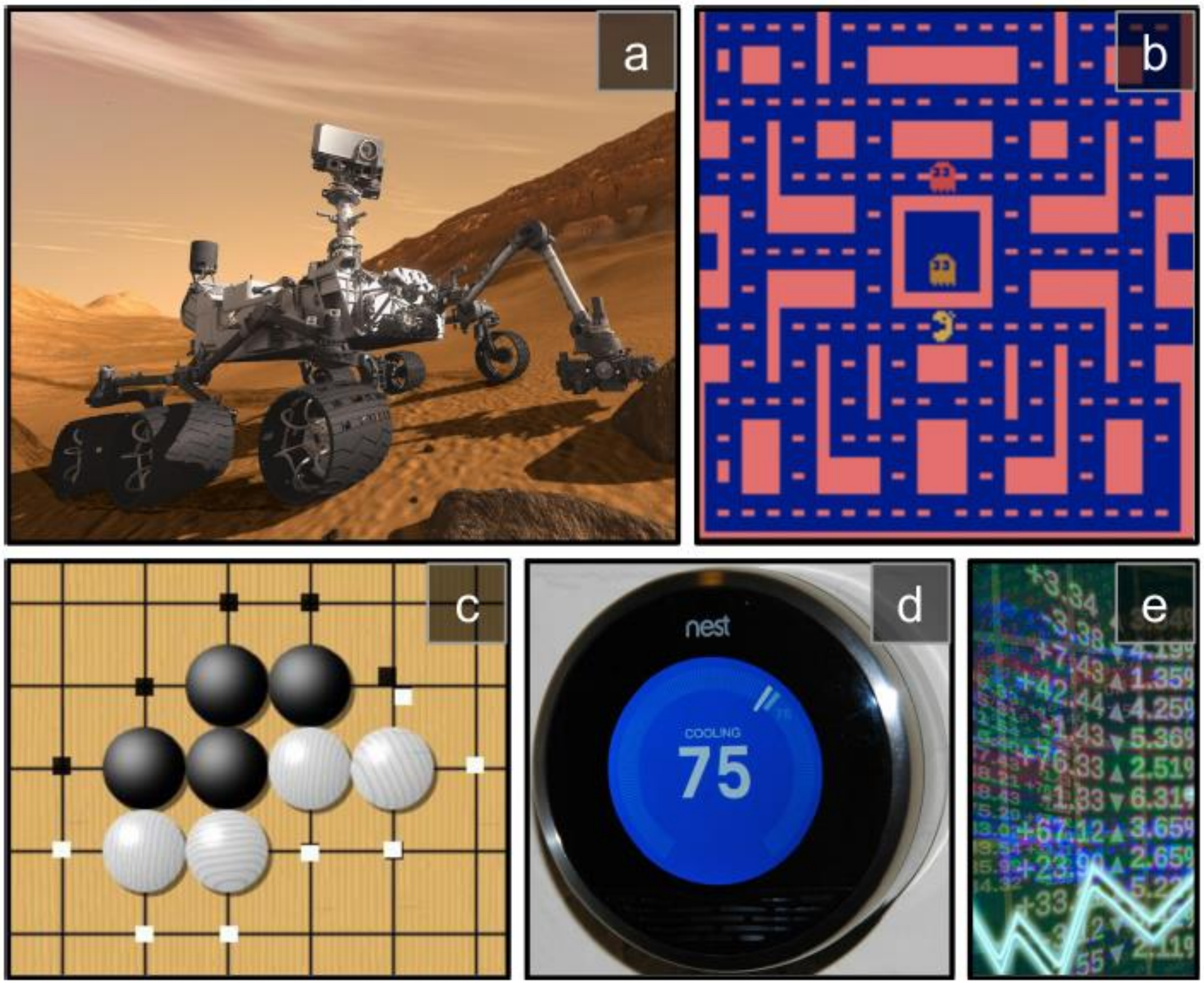
- RL started in **1950s**
- **1992**: IBM's TD-Gammon, a Backgammon playing program.
- **2013**: DeepMind demonstrated a system that learns to play Atari games from scratch.
- Use **deep learning** with raw pixels as inputs and without any prior knowledge of the rules of the games.
- **2014**: Google bought DeepMind for $500M.
- **2016**: AlphaGo beats Lee Sedol.



1. Introduction – Definition

- In Reinforcement Learning, a software **agent** makes **observations** and takes **actions** within an **environment**, and in return it receives **rewards**.
- Its objective is to learn to act in a way that will maximize its expected long-term rewards.
- In short, the agent acts in the environment and learns by trial and error to maximize its **pleasure** and minimize its **pain**.

1. Introduction – Examples



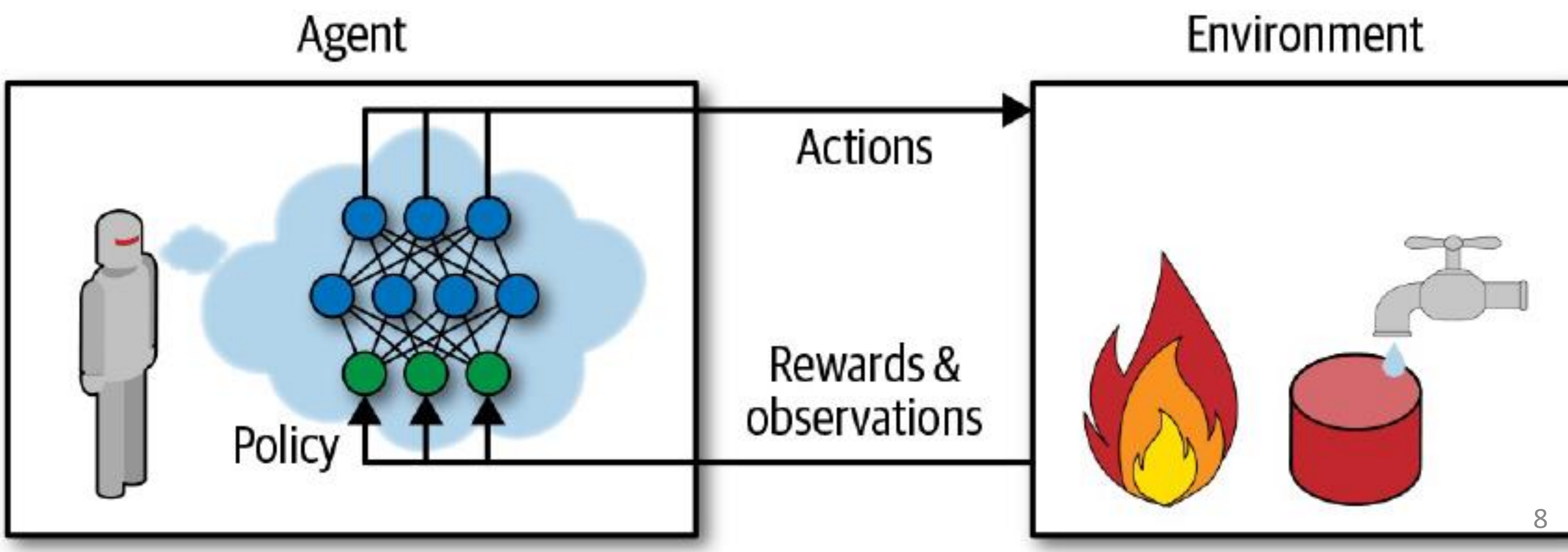
(a) robotics
(b) Ms. Pac-Man
(c) Go player
(d) thermostat
(e) automatic trader

Outline

1. Introduction
2. Policy Search
3. OpenAI Gym
4. Neural Network Policies
5. The Credit Assignment Problem
6. Q-Learning
7. Exercises

2. Policy Search

- The algorithm used by the software agent to determine its actions is called its **policy**.
- The policy can be **deterministic** or **stochastic**.
- **Policy search techniques**: Brute force, Genetic algorithm, Policy Gradient (PG), Q-Learning.



Outline

1. Introduction
2. Policy Search
3. OpenAI Gym
4. Neural Network Policies
5. The Credit Assignment Problem
6. Q-Learning
7. Exercises

3. OpenAI Gym

- OpenAI Gym is a toolkit that provides **simulated environments** (Atari games, board games, 2D and 3D physical simulations, …).
- OpenAI is a nonprofit AI research company funded in part by Elon Musk. Got $1 billion investment from Microsoft.

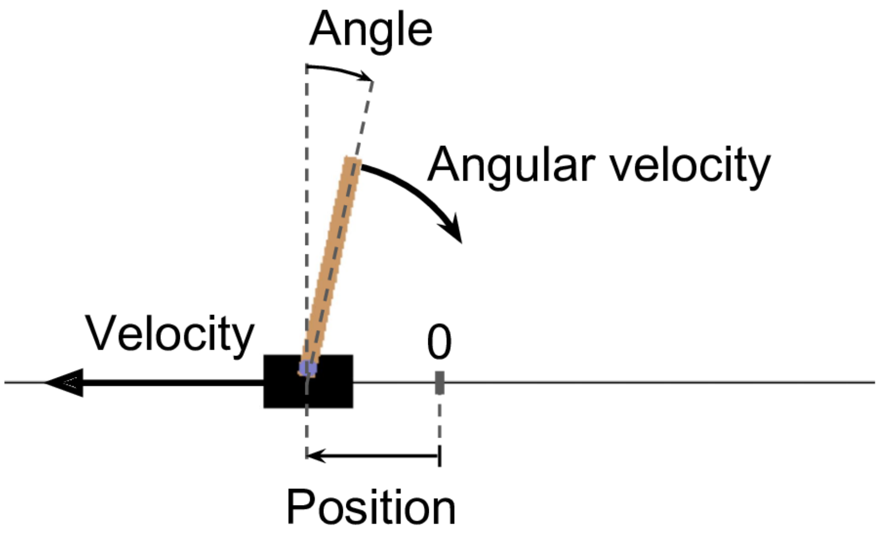
```
$ pip3 install --upgrade gym
```

Cart position, cart speed, pole angle, pole velocity

```
>>> import gym
>>> env = gym.make("CartPole-v1")
>>> obs = env.reset()
>>> obs
array([-0.012586, -0.001566, 0.042077, -0.001805])
```

3. OpenAI Gym

```
>>> env.render()
```



- **render()** can also return the rendered image as a NumPy array.

```
>>> img = env.render(mode="rgb_array")
>>> img.shape # height, width, channels (3 = RGB)
(800, 1200, 3)
```

3. Balancing the pole

```
>>> env.action_space
Discrete(2)
```

The possible actions are integers 0 and 1, which represent accelerating left (0) or right (1).

```
>>> action = 1 # accelerate right
>>> obs, reward, done, info = env.step(action)
>>> obs
array([-0.012617, 0.192928, 0.042041, -0.280921])
>>> reward
1.0
>>> done
False
>>> info
{}
```

3. Balancing the pole

```
def basic_policy(obs):
        angle = obs[2]
        return 0 if angle < 0 else 1

totals = []
for episode in range(500):
    episode_rewards = 0
    obs = env.reset()
    for step in range(200):
        action = basic_policy(obs)
        obs, reward, done, info = env.step(action)
        episode_rewards += reward
        if done:
             break
    totals.append(episode_rewards)
```

Accelerates left when the pole is leaning left and accelerates right when the pole is leaning right.

3. Balancing the pole

- Even with 500 tries, this policy never managed to keep the pole upright for more than 68 consecutive steps.

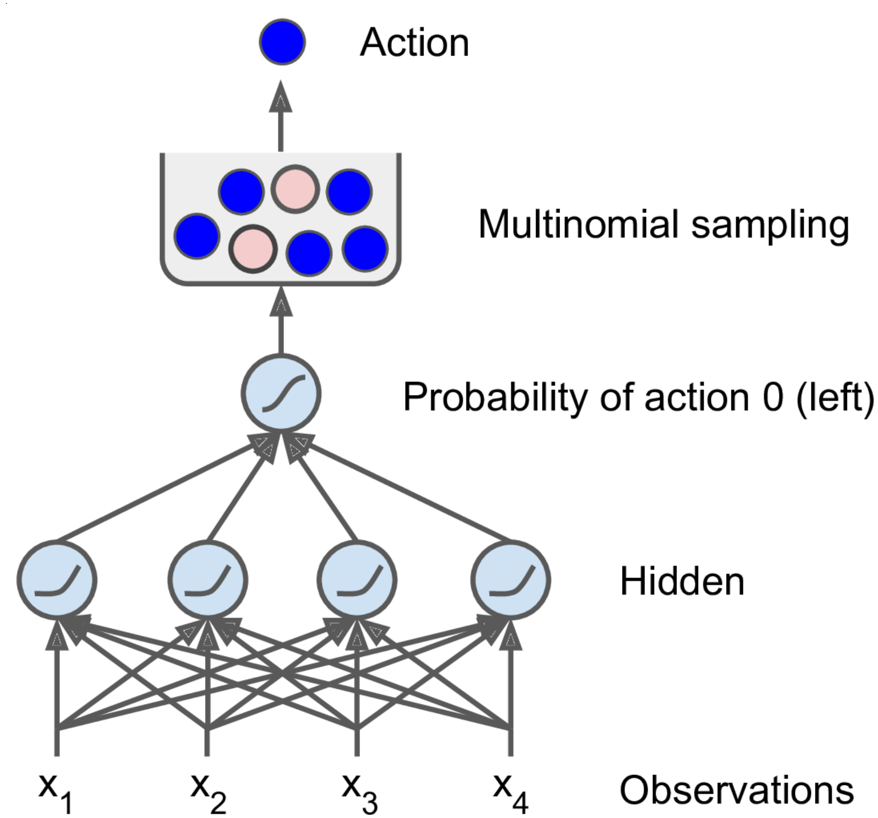
```
>>> import numpy as np
>>> np.mean(totals), np.std(totals), np.min(totals),
              np.max(totals)
(41.718, 8.858356280936096, 24.0, 68.0)
```

Outline

1. Introduction
2. Policy Search
3. OpenAI Gym
4. Neural Network Policies
5. The Credit Assignment Problem
6. Q-Learning
7. Exercises

4. Neural Network Policies

- Takes an observation as input, and outputs the probability for each action
- We select an action randomly, according to the estimated probabilities.
- Explore and exploit



4. Neural Network Policy in Keras

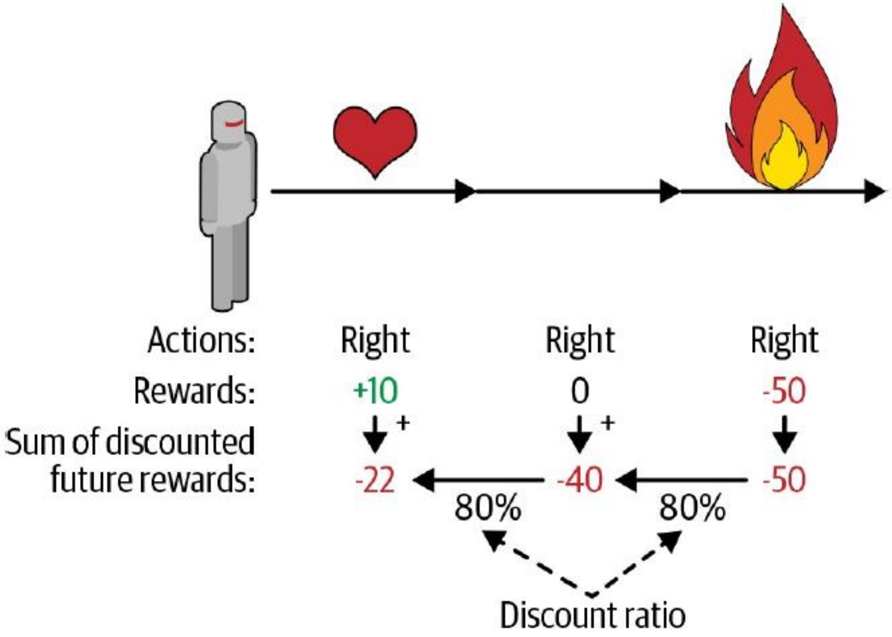
```
# Building a polity network is easy
import tensorflow as tf
from tensorflow import keras

n_inputs = 4 # == env.observation_space.shape[0]

model = keras.models.Sequential([
      keras.layers.Dense(5, activation="elu",
          input_shape=[n_inputs]),
      keras.layers.Dense(1, activation="sigmoid"),
])
# Training it is something else
```

Outline

1. Introduction
2. Policy Search
3. OpenAI Gym
4. Neural Network Policies
5. The Credit Assignment Problem
6. Q-Learning
7. Exercises

5. The Credit Assignment Problem

- Rewards are typically **sparse** and **delayed**.
- **Credit assignment problem**: when the agent gets a reward, it is hard for it to know which actions should get credited (or blamed) for it.
- Evaluate an action based on the sum of all the rewards that come after it, usually applying a **discount rate** γ at each step.



Outline

1. Introduction
2. Policy Search
3. OpenAI Gym
4. Neural Network Policies
5. The Credit Assignment Problem
6. Q-Learning
7. Exercises

6. Q-Learning

- Reference: Keon Kim, Deep Q-Learning with Keras and Gym, https://keon.io/deep-q-learning/
- Deep reinforcement learning (deep Q-learning) example to play a CartPole game using Keras and Gym.
- Google's DeepMind published Playing Atari with Deep Reinforcement Learning where they introduced the algorithm **Deep Q Network** (DQN) in 2013.
- In **DQN**, the **quality function** Q is used to approximate the reward based on a state. $Q(s,a)$ calculates the expected future value from state s and action a.
- A neural network is used to approximate the reward based on the state.

6. Q-Learning

- Carry out an action *a*, and observe the reward r and resulting new state *s'*.
- Calculate the maximum target *Q* and then discount it so that the future reward is worth less than immediate reward by γ.
- Add the current reward to the discounted future reward to get the target value.
- Subtracting our current prediction from the target gives the loss.
- Squaring this value allows us to punish the large loss value more and treat the negative values same as the positive values.

$$loss = \Big(\underbrace{\overset{\text{Reward}}{r} + \overset{\text{Decay Rate}}{\gamma} \max_{a'} \hat{Q}(s', a')}_{\text{Target}} - \underbrace{Q(s, a)}_{\text{Prediction}} \Big)^2$$

6. DQN – Imports and Definitions

```
import random
import gym
import numpy as np
from collections import deque
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam

EPISODES = 5000
```

6. DQN – Agent Class (1/4)

```
class DQNAgent:
    def __init__(self, state_size, action_size):
        self.state_size = state_size
        self.action_size = action_size
        self.memory = deque(maxlen=2000)
        self.gamma = 0.95    # discount rate
        self.epsilon = 1.0  # exploration rate
        self.epsilon_min = 0.01 # min exploration rate
        self.epsilon_decay = 0.995
        self.learning_rate = 0.001
        self.model = self._build_model()
```

6. DQN – Agent Class (2/4)

```
def _build_model(self):
    model = Sequential()
    model.add(Dense(24, input_dim=self.state_size,
          activation='relu'))
    model.add(Dense(24, activation='relu'))
    model.add(Dense(self.action_size,
          activation='linear'))
    model.compile(loss='mse',
          optimizer=Adam(lr=self.learning_rate))
    return model
```

4

2

6. DQN – Agent Class (3/4)

```
def remember(self, state, action, reward,
        next_state, done):
    # Queue of previous experiences to re-train
        the model
    self.memory.append((state, action, reward,
        next_state, done))


def act(self, state):
    # Returns an action randomly or from the model
    if np.random.rand() <= self.epsilon:
        return random.randrange(self.action_size)
    act_values = self.model.predict(state)
    return np.argmax(act_values[0])
```

6. DQN – Agent Class (4/4)

```
def replay(self, batch_size):
    minibatch = random.sample(self.memory, batch_size)
    for state, action, reward, next_state, done in
     minibatch:
        target = reward
        if not done:
            target = (reward + self.gamma * np.max(
               self.model.predict(next_state)[0]))
        target_f = self.model.predict(state)
        target_f[0][action] = target
        self.model.fit(state, target_f, epochs=1,
         verbose=0)
    if self.epsilon > self.epsilon_min:
        self.epsilon *= self.epsilon_decay
```

`Replay()` trains the neural net with experiences in the memory

$$loss = \left(r + \gamma \max_{a'} \hat{Q}(s', a') - Q(s, a) \right)^2$$

Learn to predict the reward

6. DQN – Setup

```
if __name__ == "__main__":
    env = gym.make('CartPole-v1')
    state_size = env.observation_space.shape[0] # 4
    action_size = env.action_space.n            # 2
    agent = DQNAgent(state_size, action_size)
    done = False
    batch_size = 32
```

6. DQN – Training

```
for e in range(EPISODES):
    state = env.reset()
    state = np.reshape(state, [1, state_size])
    for time in range(5000):
        action = agent.act(state)
        next_state, reward, done, _ = env.step(action)
        reward = reward if not done else -10
        next_state = np.reshape(next_state, [1, state_size])
        agent.remember(state, action, reward, next_state,
               done)
        state = next_state
        if done:
            print("episode: {}/{}, score: {}"
                   .format(e, EPISODES, time))
            break
        if len(agent.memory) > batch_size:
            agent.replay(batch_size)
```

6. DQN – Results

```
episode: 1/5000, score: 27
episode: 2/5000, score: 11
episode: 3/5000, score: 34
episode: 4/5000, score: 33
episode: 5/5000, score: 8
episode: 6/5000, score: 22
episode: 7/5000, score: 47
episode: 8/5000, score: 22
episode: 9/5000, score: 54
episode: 10/5000, score: 16
```

```
episode: 284/5000, score: 1331
episode: 285/5000, score: 124
episode: 286/5000, score: 259
episode: 287/5000, score: 138
episode: 288/5000, score: 170
episode: 289/5000, score: 13
episode: 290/5000, score: 365
episode: 291/5000, score: 1499
episode: 292/5000, score: 274
episode: 293/5000, score: 498
episode: 294/5000, score: 529
episode: 295/5000, score: 284
episode: 296/5000, score: 1355
episode: 297/5000, score: 911
episode: 298/5000, score: 1414
```

Exercises

From Chapter 18, solve exercises:

- 1 – 6

Summary

1. Introduction
2. Policy Search
3. OpenAI Gym
4. Neural Network Policies
5. The Credit Assignment Problem
6. Q-Learning
7. Exercises