# LAPORAN TUGAS 3

# MACHINE LEARNING

*Q-Learning*


Disusun Oleh:
Aditya Alif Nugraha
1301154183
IF-39-01

PRODI S1 TEKNIK INFORMATIKA
FAKULTAS INFORMATIKA
UNIVERSITAS TELKOM
BANDUNG
2018

# Analisa Masalah

Pada tugas 3 ini, mahasiswa diuji kemampuannya untuk menganalisa, mendesain, dan mengimplementasi salah satu Metode *Reinforcement Learning* yaitu *Q-Learning.* Metode tersebut memungkinkan sebuah agen dapat mempelajari *environment*-nya tanpa sebelumnya mengetahui kondisi keseluruhan. *Environment* berisi *rewards* yang akan didapatkan agen setiap melakukan langkah.

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 10 | -1 | -3 | -5 | -1 | -3 | -3 | -5 | -5 | -1 | 100 |
| 9 | -2 | -1 | -1 | -4 | -2 | -5 | -3 | -5 | -5 | -5 |
| 8 | -3 | -4 | -4 | -1 | -3 | -5 | -5 | -4 | -3 | -5 |
| 7 | -3 | -5 | -2 | -5 | -1 | -4 | -5 | -1 | -3 | -4 |
| 6 | -4 | -3 | -3 | -2 | -1 | -1 | -1 | -4 | -3 | -4 |
| 5 | -4 | -2 | -5 | -2 | -4 | -5 | -1 | -2 | -2 | -4 |
| 4 | -4 | -3 | -2 | -3 | -1 | -3 | -4 | -3 | -1 | -3 |
| 3 | -4 | -2 | -5 | -4 | -1 | -4 | -5 | -5 | -2 | -4 |
| 2 | -2 | -1 | -1 | -4 | -1 | -3 | -5 | -1 | -4 | -1 |
| 1 | -5 | -3 | -1 | -2 | -4 | -3 | -5 | -2 | -2 | -2 |
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

Figure 1: Sebuah *grid world* ukuran 10 x 10, di mana angka-angka dalam kotak menyatakan *reward*. *Agent* berada di posisi *Start* (1,1) dan *Goal* di posisi (10,10)

Pada tugas ini, diberikan data berupa *environment* yang akan digunakan untuk melatih agen. Mahasiswa ditugaskan untuk melatih agen berjalan dari titik awal (1,1) sampai (10,10) untuk mendapatkan jumlah *reward* yang sebesar-besarnya. Aksi yang diperbolehkan yaitu bergerak ke arah N (*North*), E (*East*), W (*West*), S (*South*). Untuk melatih agen, akan digunakan Q *table* dan R *table* yang berukuran 100x4 yang merepresentasikan state dan action. R *table* berisi *action* dan *reward* yang akan didapatkan agen. Dan hasil dari pelatihan agen berupa Q table yang terupdate setelah menjalani beberapa episode. Setelah berhasil melatih agen, akan didapatkan jalur berdasarkan Q table yang akan digunakan untuk menjalani *environment*.

# Desain

## Algoritma Q-Learning

Algoritma Q-Learning untuk kasus *Grid World*:

1. Tentukan parameter gamma, jumlah episode, dan environment dalam bentuk matriks R
2. Inisialisasi Q table dan R table dengan representasi masing-masing (untuk kasus ini, digunakan matriks berukuran 100x4 (100 state, 4 aksi)
3. Lakukan sebanyak episode:
    a. Pilih random initial state
    b. Lakukan selama agen belum mencapai goal state
        i. Dapatkan semua aksi yang mungkin dijalankan
        ii. Pilih acak salah satu dari aksi yang memungkinkan
        iii. Hitung: Q(state, action) = R(state, action) + gamma * Max[Q(next state, all actions)]
        iv. Ubah *current state* menjadi *state* setelah dilakukan aksi yang terpilih
    c. End do
4. End for

## Penentuan Nilai Gamma

Rentang nilai gamma yaitu 0 <= gamma <= 1. Semakin gamma mendekati 0 maka agen akan bergantung pada *immediate reward*. Jika gamma mendekati 1, maka agen akan bergantung pada *future reward*.

Pada kasus ini, dipilih nilai **gamma = 0.9**. Nilai tersebut dipilih agar agen lebih mementingkan reward yang didapatkan jika melakukan aksi (*future reward*) dibandingkan reward pada state tersebut (*immediate reward*).

## Observasi Jumlah Episode

Agen dikatakan berhasil di*training* jika berhasil berjalan dari state (1,1) ke (10,10) dengan reward maksimal. Jika jumlah episode terlalu sedikit dapat menyebabkan agen tidak sampai ke goal state dan malah mondar-mandir di *environment*. Jika terlalu banyak episode, maka akan memakan waktu running yang lama.

Pada kasus ini, setelah menjalani **50 episode**, agen berhasil mencapai goal state dengan reward yang konstan/tetap. Reward yang didapatkan yaitu **65**. **1 episode** dimulai saat agen berjalan dari *random initial state* dan berakhir saat agen mencapai *goal state*.

## Pencarian Jalur Terbaik

Setelah menjalani beberapa episode, maka matriks/tabel Q akan terupdate nilainya. Dari nilai tersebut akan diambil nilai maksimal pada action di current state. Setelah melakukan *action*, maka agen akan menuju ke state yang berikutnya. Hal tersebut dilakukan terus menerus hingga agen mencapai *goal state*.

Algoritma:

1. Tentukan *initial state* dan *goal state*
2. Inisialisasi variable current state = initial state
3. Inisialisasi variable rewards = 0
4. Inisialisasi variable jalur = list()
5. While current state != goal state do
    a. Pilih aksi dengan reward tertinggi berdasarkan tabel Q
    b. Pindah ke state baru setelah melakukan aksi
    c. Jumlahkan variable rewards dengan reward yang didapatkan pada current state di *environment*
    d. Tambahkan jalur yang dipilih ke variable jalur
6. End do

## Representasi Tabel Q

Q Table

| | 0 | | | | | | | | | | ... | 9 | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| **E** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **N** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **S** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **W** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

4 rows × 100 columns

NB: Table di *transpose* agar memudahkan dalam *screenshot.*

## Representasi Tabel R

R Table

| | 0 | | | | | | | | | | ... | 9 | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| **E** | -3.0 | -1.0 | -2.0 | -4.0 | -3.0 | -5.0 | -2.0 | -2.0 | -2.0 | NaN | ... | -3.0 | -5.0 | -1.0 | -3.0 | -3.0 | -5.0 | -5.0 | -1.0 | 100.0 | NaN |
| **N** | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | -2.0 | -1.0 | -1.0 | -4.0 | -2.0 | -5.0 | -3.0 | -5.0 | -5.0 | -5.0 |
| **S** | -2.0 | -1.0 | -1.0 | -4.0 | -1.0 | -3.0 | -5.0 | -1.0 | -4.0 | -1.0 | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| **W** | NaN | -5.0 | -3.0 | -1.0 | -2.0 | -4.0 | -3.0 | -5.0 | -2.0 | -2.0 | ... | NaN | -1.0 | -3.0 | -5.0 | -1.0 | -3.0 | -3.0 | -5.0 | -5.0 | -1.0 |

4 rows × 100 columns

NB: NaN menyatakan aksi pada state tersebut tidak dapat dilakukan. Tabel di *transpose* agar memudahkan dalam *screenshot.* Lihat file “R Table.xlsx” untuk hasil lengkap.

# Evaluasi Hasil Eksperimen

## Hasil Matriks Q

Setelah menjalani 50 episode, hasil matriks/tabel Q menjadi terupdate. Reward yang mendekati goal state akan menjadi lebih besar dibandingkan yang jauh dari goal state. Hal tersebut menandakan bahwa semakin mendekati goal state, maka reward yang didapatkan semakin besar.

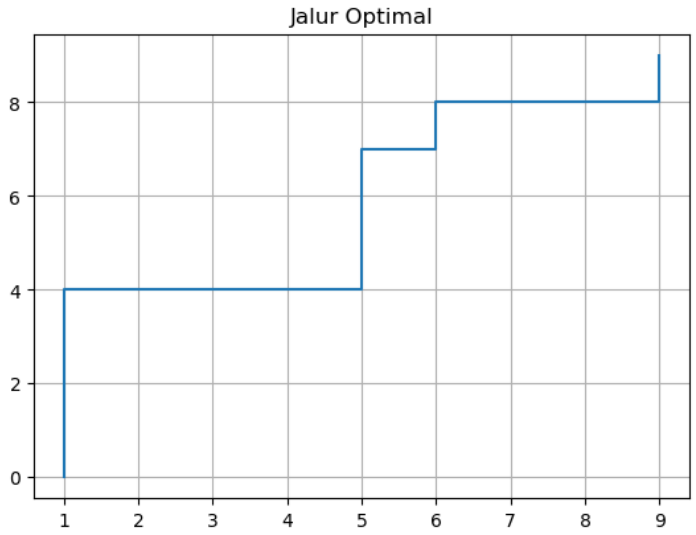
Q Table

| | | E | N | S | W |
|---|---|---|---|---|---|
| **0** | **0** | -0.007896 | 0.0 | 0.992104 | 0.000000 |
| | **1** | 3.324560 | 0.0 | 3.324560 | -4.107107 |
| | **2** | 3.805066 | 0.0 | 4.805066 | -0.007896 |
| | **3** | 6.450074 | 0.0 | 6.450074 | 3.324560 |
| | **4** | 4.925063 | 0.0 | 11.611193 | 3.805066 |

NB: silahkan buka file “Q Table.xlsx” untuk hasil lengkap.

## Jalur dan Total Rewards

```
============== HASIL TRAINING Q-LEARNING ==============
Jalur:  (1, 0) (1, 1) (1, 2) (1, 3) (1, 4) (2, 4) (3, 4) (4, 4) (5, 4) (5, 5) (5, 6) (5, 7) (6, 7) (6, 8) (7, 8) (8, 8) (9, 8) (9, 9)
Total Rewards:  65
```



## Implementasi

### Main File

```
# Nama  : Aditya Alif Nugraha
# NIM   : 1301154183
# Kelas : IF-39-01

import pandas as pd
import fun
from qlearning import QLearning
import numpy as np

if __name__ == '__main__':
    env = fun.load_environment("DataTugasML3.txt")
    env = np.flip(env, axis=0)
    print(env)

    ql = QLearning(env)
    ql.fit(50)
    tracks, rewards = ql.predict()

    # print(ql.q_table)
    print("=============== HASIL TRAINING Q-LEARNING ===============")
    print("Jalur: ", *tracks)
    print("Total Rewards: ", rewards)
    fun.visualize_tracks(tracks)
```

### Q-Learning Core Class

```
# Nama  : Aditya Alif Nugraha
# NIM   : 1301154183
# Kelas : IF-39-01

import numpy as np
import fun
import random
import sys


class QLearning:
    """
    Class Q-Learning implementing the Q-Learning Algorithm for grid world.
    This code will be submitted to scikit-learn by Aditya Alif Nugraha.
    """

    def __init__(self, environment, gamma=0.9):
        """Inisialisasi variable yang dibutuhkan."""
        self.environment = environment
        self.environment_shape = environment.shape
```

```
        self.gamma = gamma
        self.q_table = self.build_q_table()
        self.r_table = self.build_r_table()
        self.initial_state = (0, 0)
        self.current_state = (0, 0)
        self.finish_state = np.unravel_index(
            np.argmax(environment, axis=None), environment.shape)

    def build_q_table(self):
        """Membangun Q table sejumlah grid dikali jumlah action untuk
menyimpan hasil learning."""
        q_table = {}
        for i in range(self.environment_shape[0]):
            for j in range(self.environment_shape[1]):
                q_table[(i, j)] = {"N": 0, "E": 0, "S": 0, "W": 0}
        return q_table

    def build_r_table(self):
        """Membangun table rewards sejumlah grid dikali jumlah action untuk
menyimpan hasil learning."""
        r_table = {}
        for i in range(self.environment_shape[0]):
            for j in range(self.environment_shape[1]):
                r_table[(i, j)] = {}

                # to North
                if i-1 < 0:
                    r_table[(i, j)]["N"] = None
                else:
                    r_table[(i, j)]["N"] = self.environment[i-1, j]

                # to East
                if j+1 >= self.environment_shape[1]:
                    r_table[(i, j)]["E"] = None
                else:
                    r_table[(i, j)]["E"] = self.environment[i, j+1]

                # to South
                if i+1 >= self.environment_shape[0]:
                    r_table[(i, j)]["S"] = None
                else:
                    r_table[(i, j)]["S"] = self.environment[i+1, j]

                # to West
                if j-1 < 0:
                    r_table[(i, j)]["W"] = None
                else:
                    r_table[(i, j)]["W"] = self.environment[i, j-1]
```

```
        return r_table

    def getNextState(self, action):
        if action == "N":
            next_state = (self.current_state[0]-1, self.current_state[1])
        elif action == "E":
            next_state = (self.current_state[0], self.current_state[1]+1)
        elif action == "S":
            next_state = (self.current_state[0]+1, self.current_state[1])
        elif action == "W":
            next_state = (self.current_state[0], self.current_state[1]-1)
        return next_state

    def update_q(self):
        """Mengupdate nilai Q di Q table pada current state."""
        action = self.select_action()
        next_state = self.getNextState(action)
        self.q_table[self.current_state][action] = self.q_formula(
            self.current_state, next_state, action)
        self.current_state = next_state

    def getPossibleAction(self):
        possible_action = []
        for action, reward in self.r_table[self.current_state].items():
            if reward != None:
                possible_action.append(action)
        return possible_action

    def select_action(self):
        """Memilih aksi yang dapat dilakukan."""
        possible_action = self.getPossibleAction()
        action = random.sample(possible_action, 1)
        return action[0]

    def q_formula(self, current_state, next_state, action):
        """Menghitung nilai q."""
        q_value = self.r_table[current_state][action] + \
            (self.gamma * max(self.q_table[next_state].values()))
        return q_value

    def fit(self, episodes):
        """Melakukan training."""
        self.current_state = (random.randint(0, 9), random.randint(0, 9))
        for _ in range(episodes):
            while self.current_state != self.finish_state:
                self.update_q()
            self.current_state = (random.randint(0, 9), random.randint(0, 9))
```

```
    def predict(self):
        """Mendapatkan solusi terbaik berupa jalur dan total rewards."""
        try:
            tracks = []
            rewards = 0
            self.current_state = (0, 0)
            while self.current_state != self.finish_state:
                q_actions = list(self.q_table[self.current_state].keys())
                q_rewards = list(self.q_table[self.current_state].values())

                action = q_actions[np.argmax(q_rewards)]
                self.current_state = self.getNextState(action)
                tracks.append(self.current_state)
                rewards += self.environment[self.current_state[0],
                                            self.current_state[1]]
            return tracks, rewards
        except:
            print("Please add more episodes!")
            sys.exit(0)
```

Additional Function

```
# Nama  : Aditya Alif Nugraha
# NIM   : 1301154183
# Kelas : IF-39-01

import numpy as np
import matplotlib.pyplot as plt


def load_environment(filename):
    """Meload file environment."""
    env = []
    with open(filename) as f:
        for line in f:
            env.append(line.split())
    return np.array(env, dtype=int)


def visualize_tracks(tracks):
    tracks = np.array(tracks)
    plt.plot(tracks[:, 0], tracks[:, 1])
    plt.axis("tight")
    plt.title("Jalur Optimal")
    plt.grid(True)
    plt.show()
```