**LAPORAN TUGAS 3**

**MACHINE LEARNING**

*Q-Learning*



Disusun Oleh:

Aditya Alif Nugraha

1301154183

IF 39-01

PRODI S1 TEKNIK INFORMATIKA

FAKULTAS INFORMATIKA

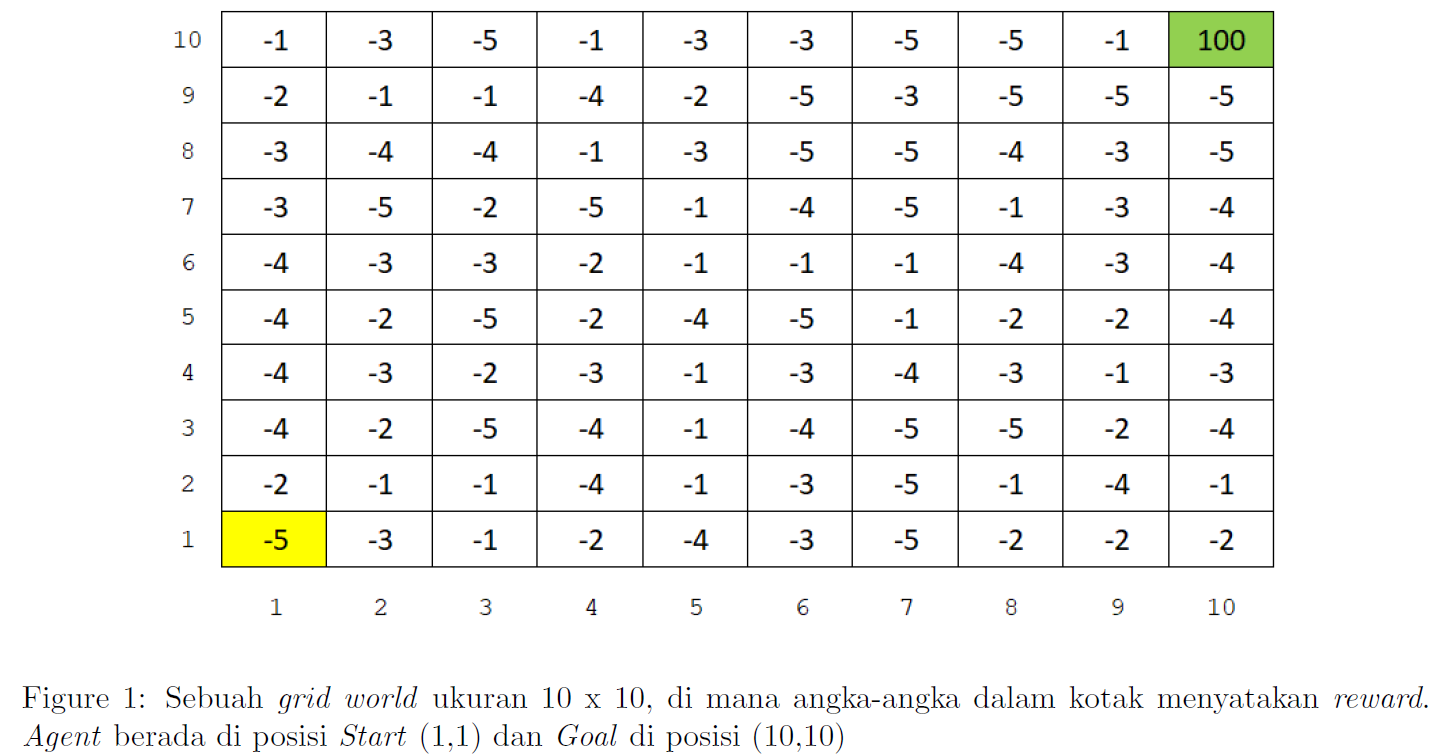
UNIVERSITAS TELKOM

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.



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) dengan 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*.

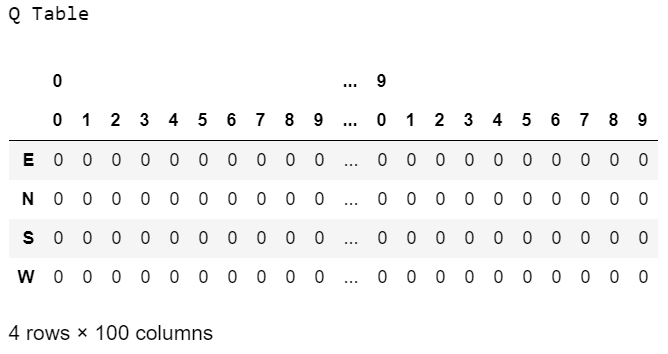
# Desain

## Algoritma Q-Learning

Algoritma Q-Learning untuk kasus *Grid World*:

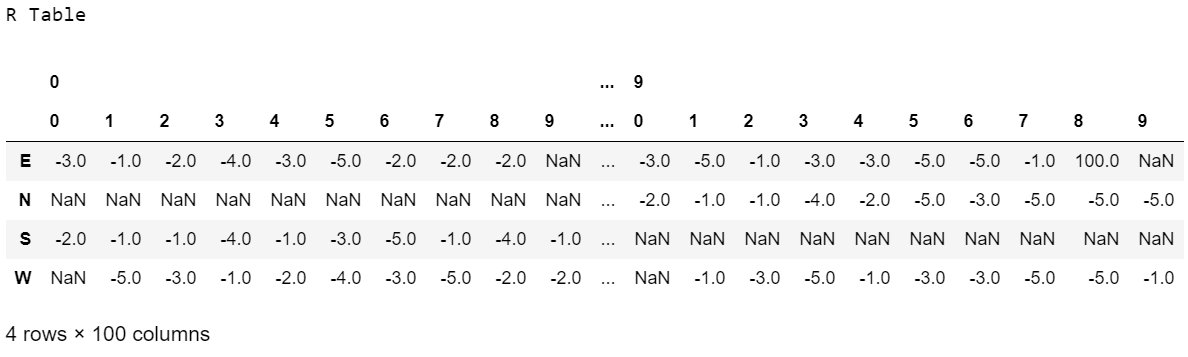
1. Tentukan parameter gamma, episode, dan environment dalam bentuk matriks R
2. Inisialisasi Q table dan R table dengan representasi masing-masing (untuk kasus ini, digunakan matriks berukuran 100x4 (100 state, 4 aksi)
3. Lakukan sebanyak episode:
   1. Pilih random initial state
   2. Lakukan selama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

## Representasi Tabel Q



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

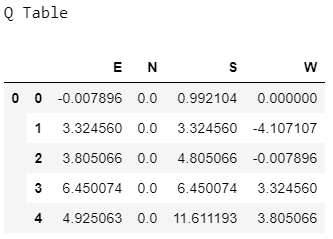
## Representasi Tabel R



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

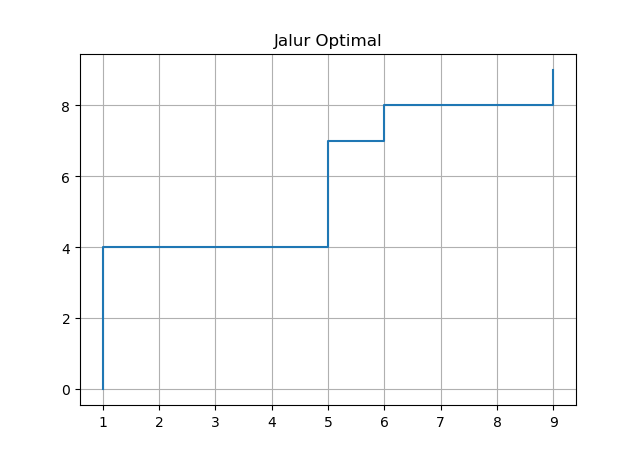
## Hasil Matriks Q



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

## Jalur dan Total Rewards





## 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.

"""

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.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 = (0, 0)

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 = []

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 = 0

for track in tracks:

rewards += self.environment[track[0], track[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()