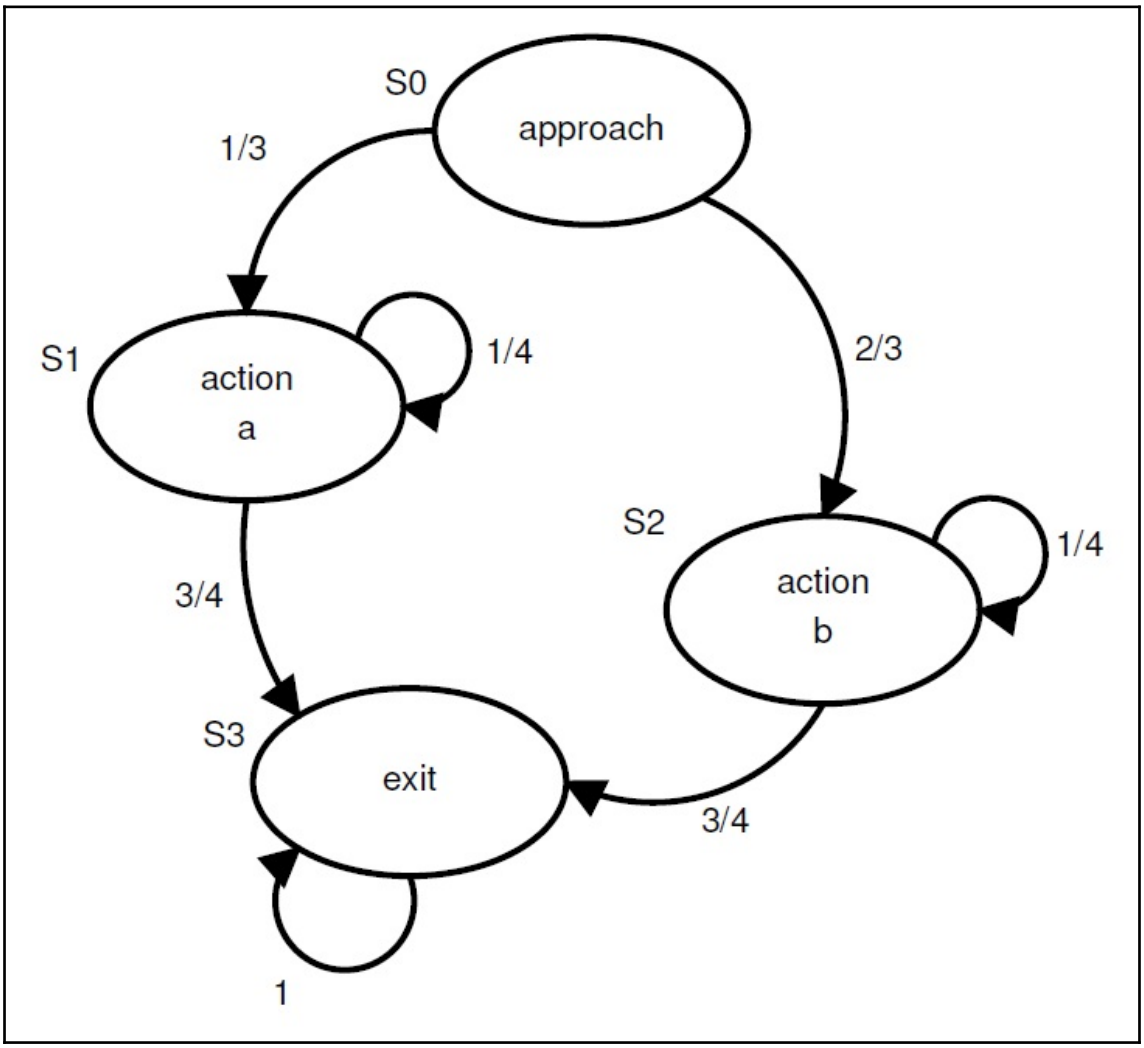
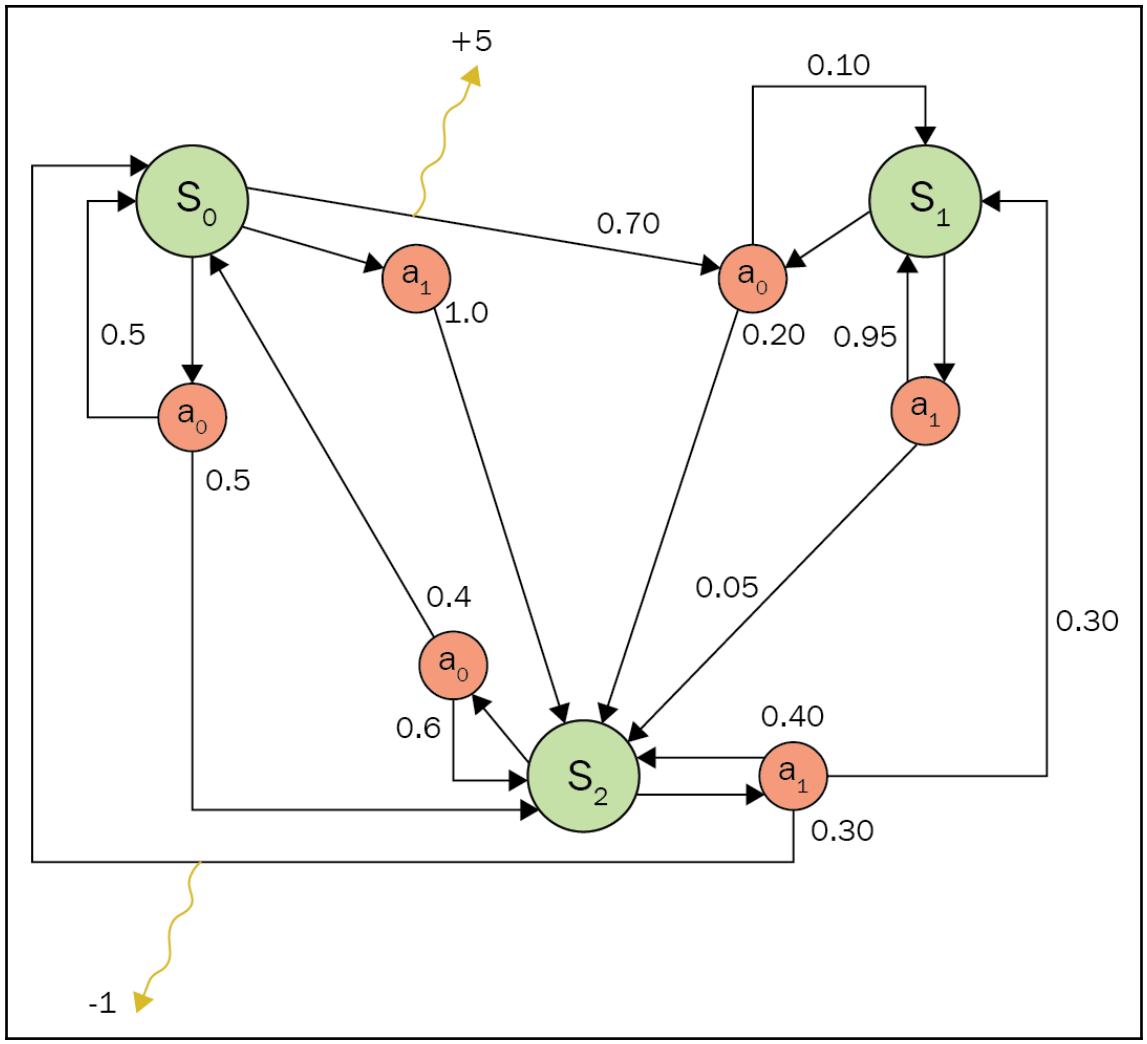
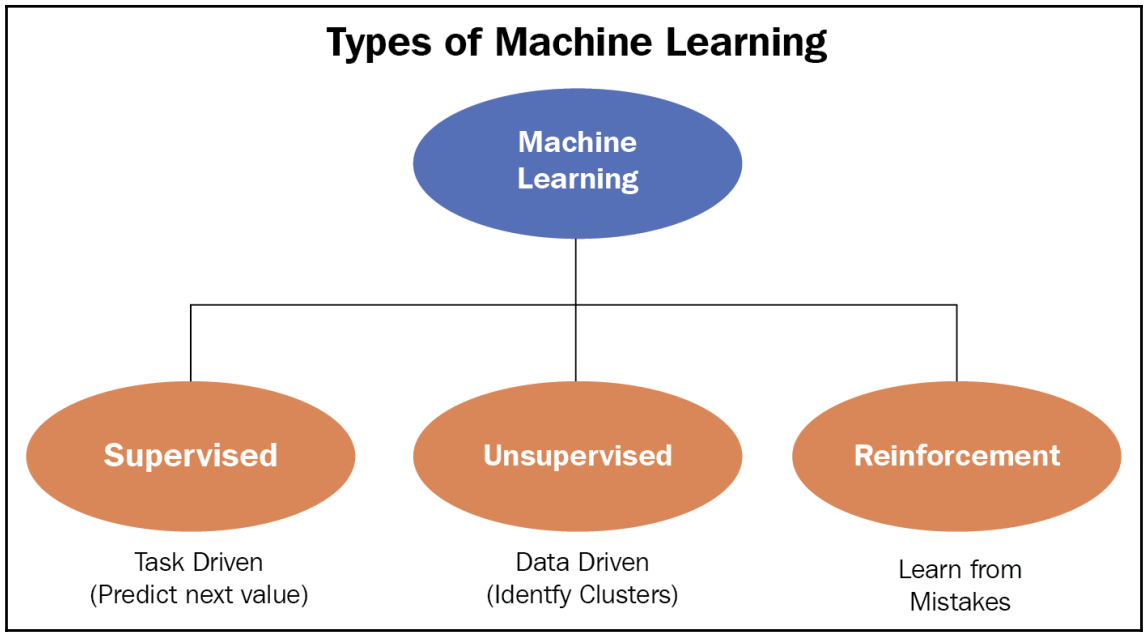
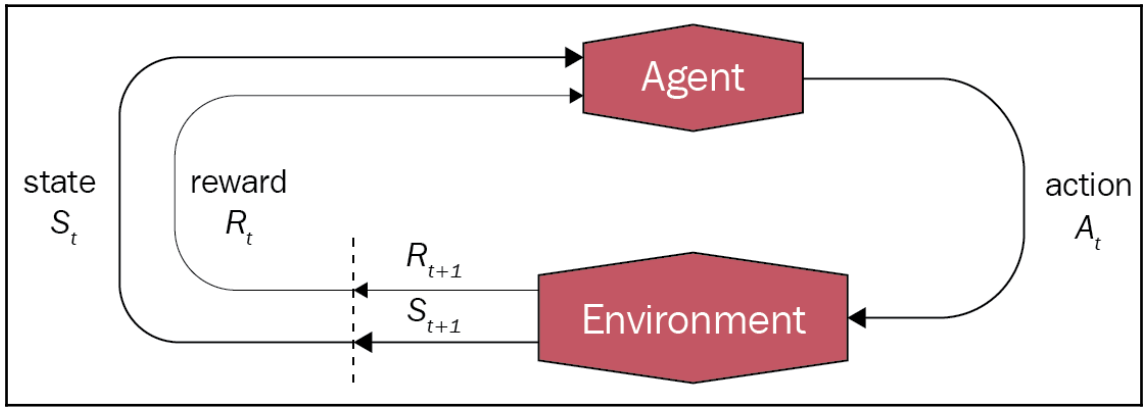
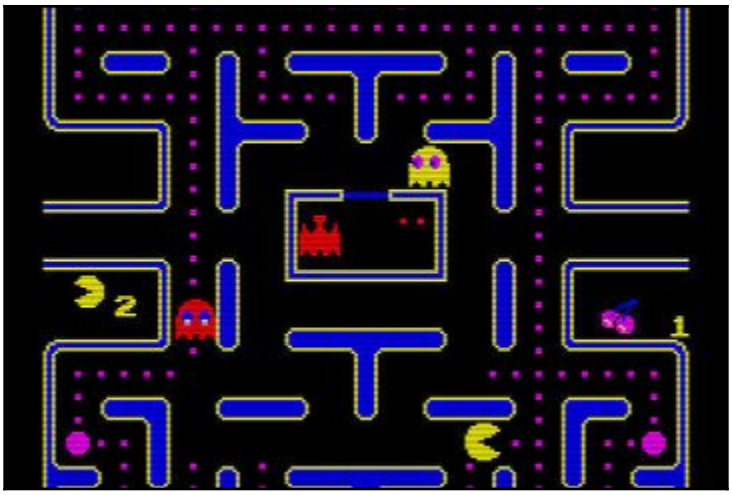
Chapter 1, Brushing Up on Reinforcement Learning Concepts




+5
0.10
S0
S1
0.70
a1
a0
1.0
0.20
0.95
0.5
a0
a1
0.5
0.4
0.05
0.30
a0
0.6
0.40
S2
a1
0.30
-1


Types of Machine Learning
Machine Learning
Supervised
Unsupervised
Reinforcement
Task Driven
(Predict next value)
Data Driven
(Identfy Clusters)
Learn from
Mistakes


Agent
state
S_t
reward
R_t
action
A_t
R_{t+1}
S_{t+1}
Environment



```
import gym
env = gym.make("Taxi-v2")
env.render()
```

```
+---------+
|R: | : :G|
| : : : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
```

```
print("Action Space {}".format(env.action_space))
print("State Space {}".format(env.observation_space))
```

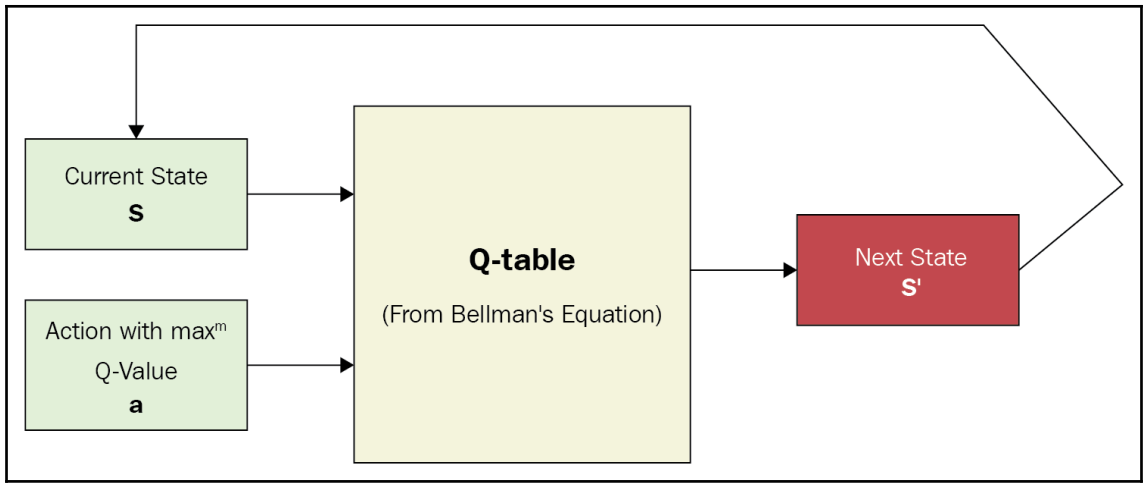
```
Action Space Discrete(6)
State Space Discrete(500)
```

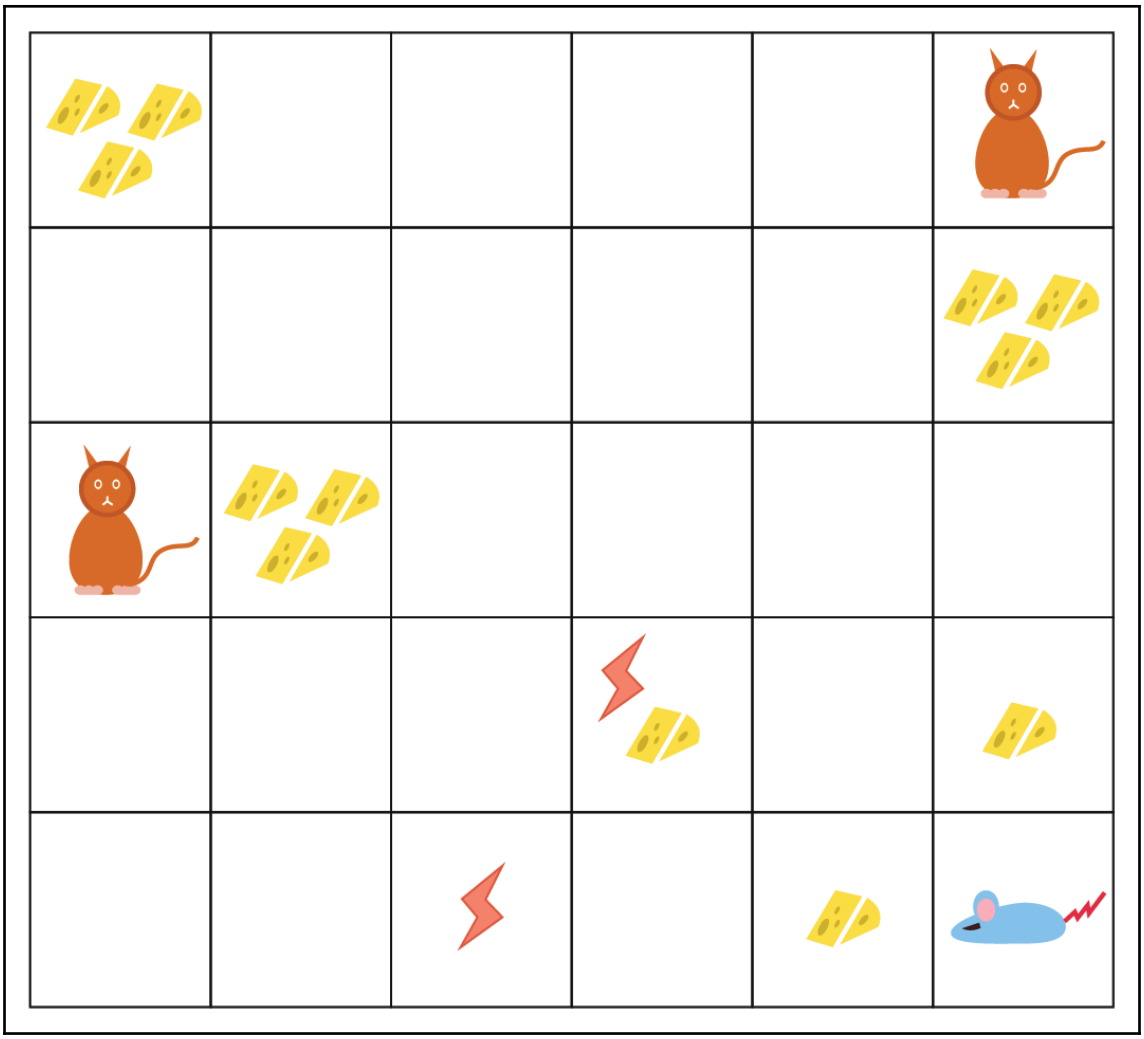
Initialized

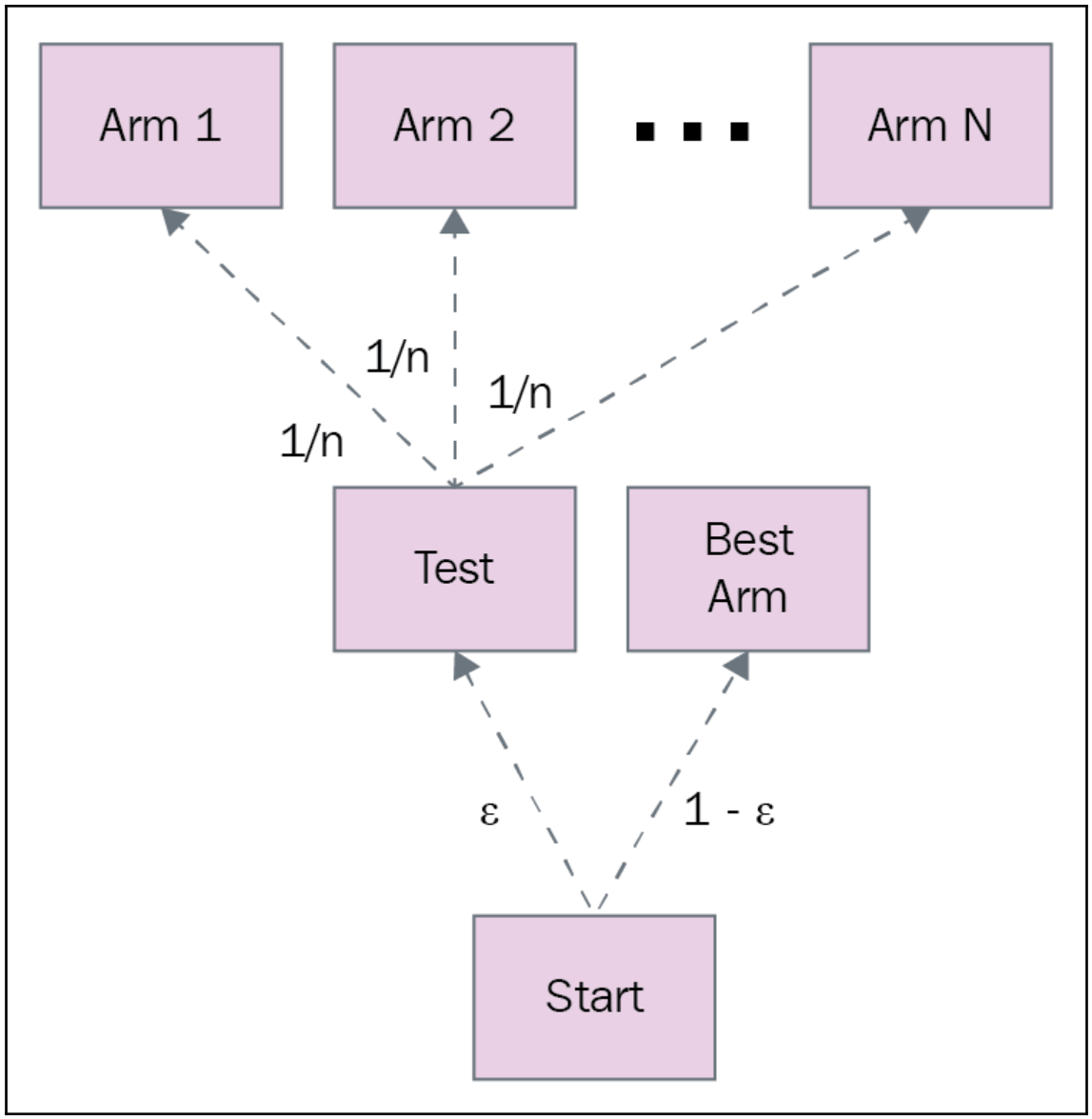
Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	327	0	0	0	0	0	0
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	499	0	0	0	0	0	0

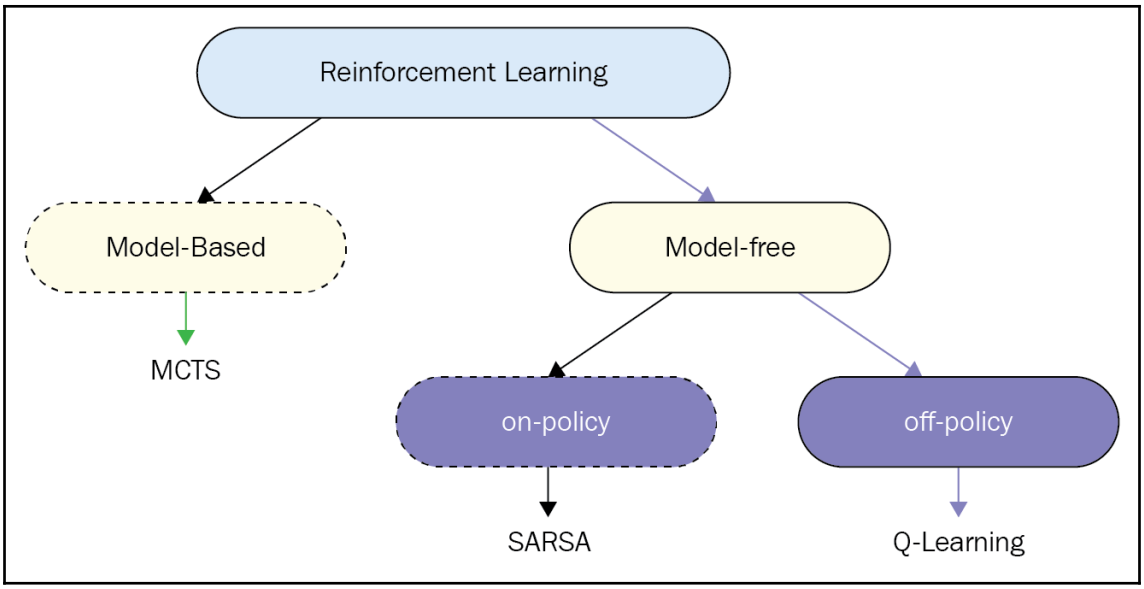
Training

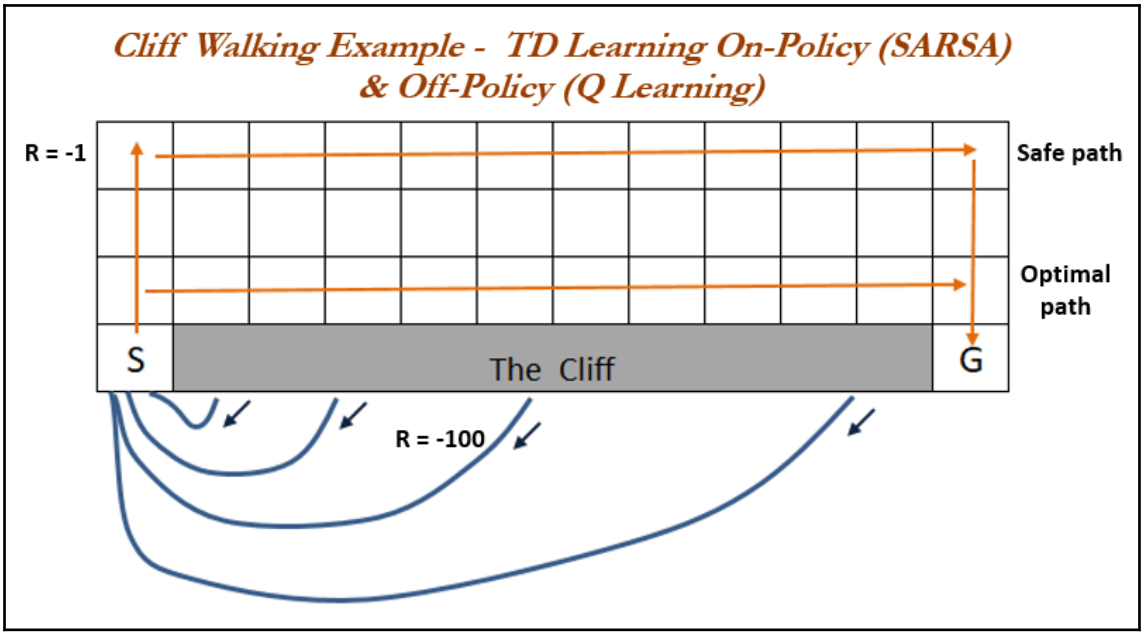
Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	328	-2.30108105	-1.97092096	-2.30357004	-2.20591839	-10.3607344	-8.5583017
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	499	9.96984239	4.02706992	12.96022777	29	3.32877873	3.38230603


Current State
S
Action with max^m
Q-Value
a
Q-table
(From Bellman's Equation)
Next State
S'

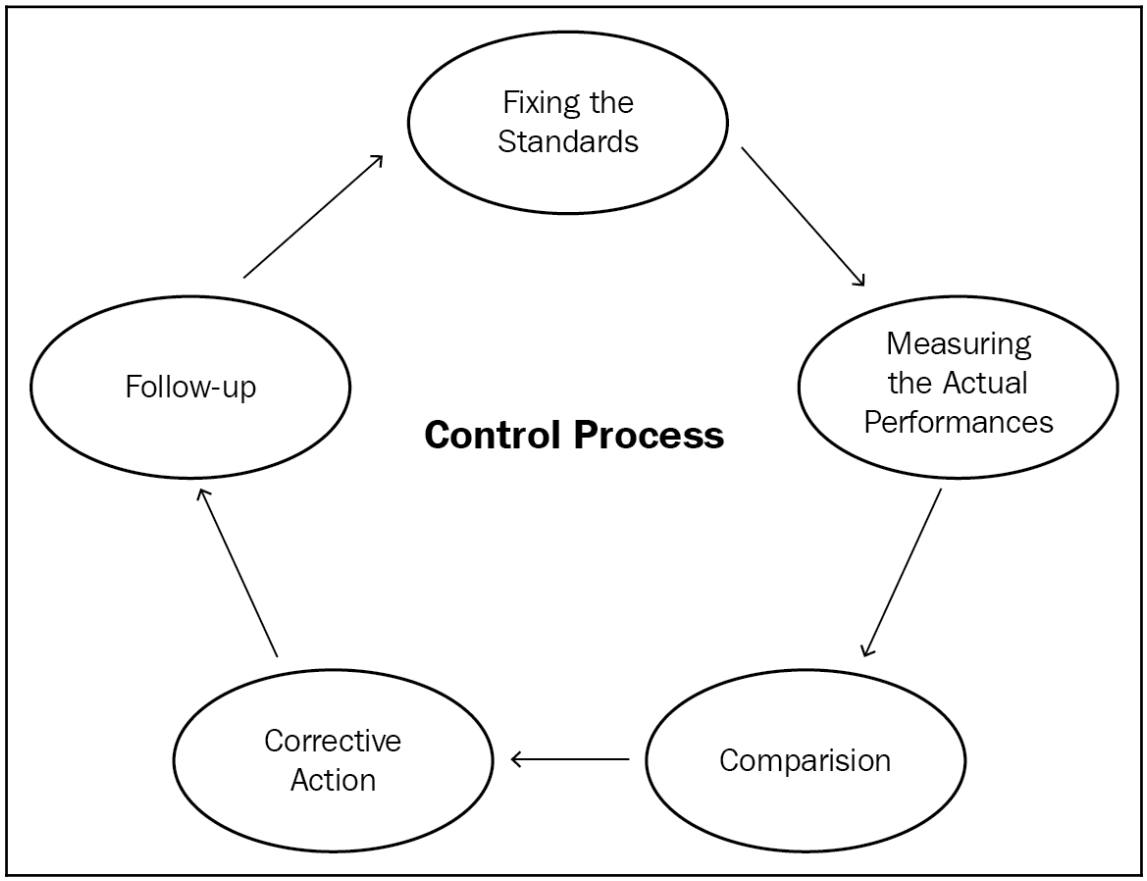


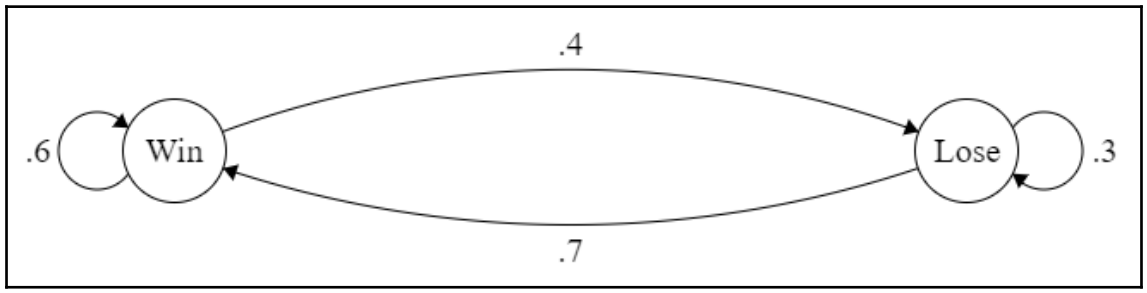

Arm 1
Arm 2
Arm N
1/n
1/n
1/n
Test
Best Arm
ε
1 - ε
Start

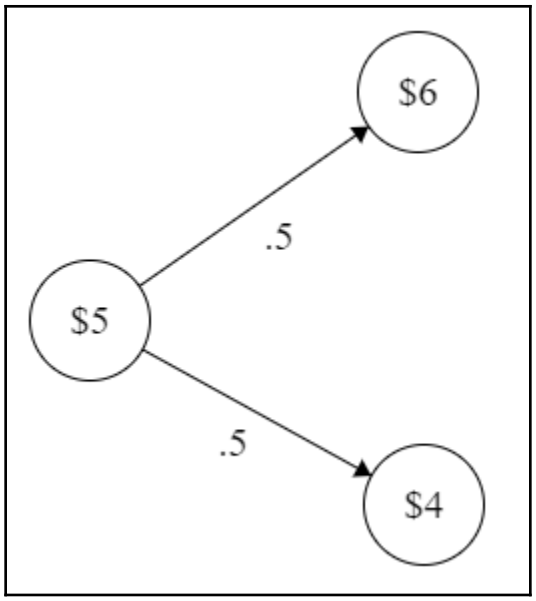

Reinforcement Learning
Model-Based
Model-free
MCTS
on-policy
off-policy
SARSA
Q-Learning

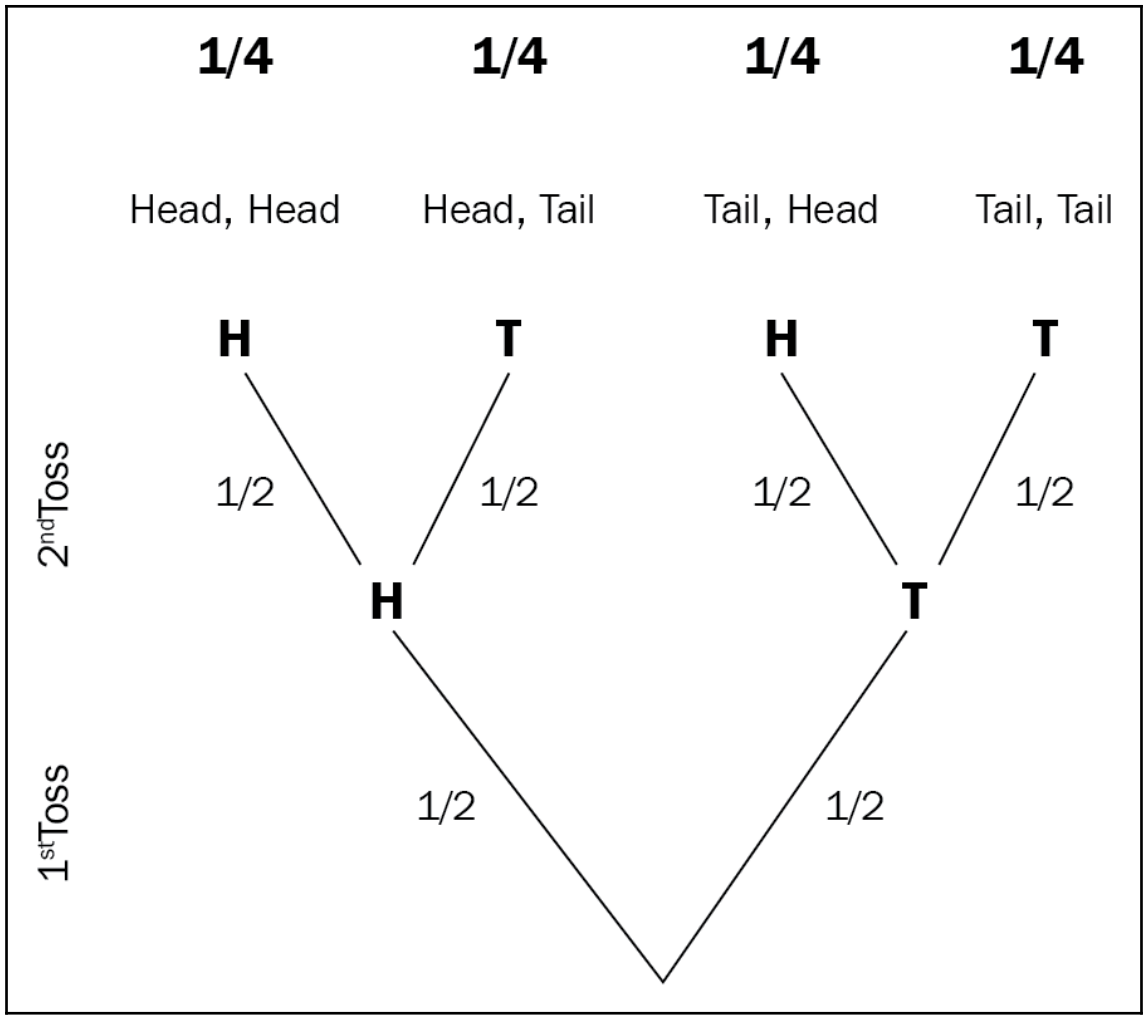

Cliff Walking Example - TD Learning On-Policy (SARSA)
& Off-Policy (Q Learning)
R = -1
Safe path
Optimal path
S
The Cliff
G
R = -100

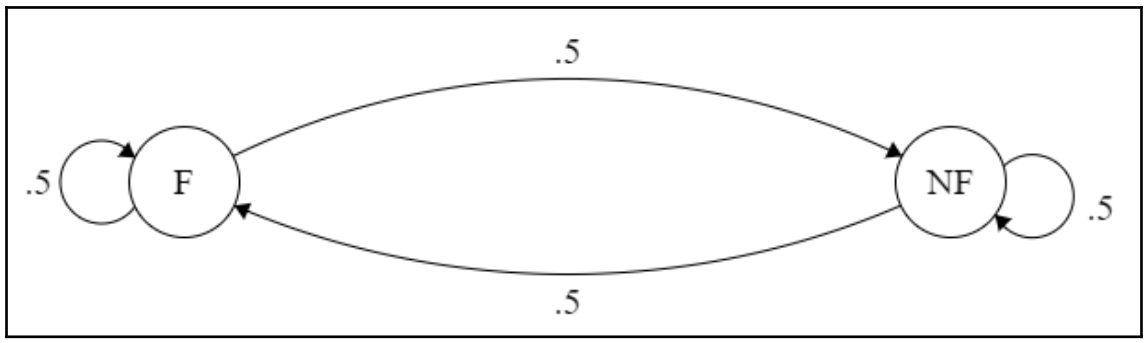
Chapter 2, Getting Started with the Q-Learning Algorithm

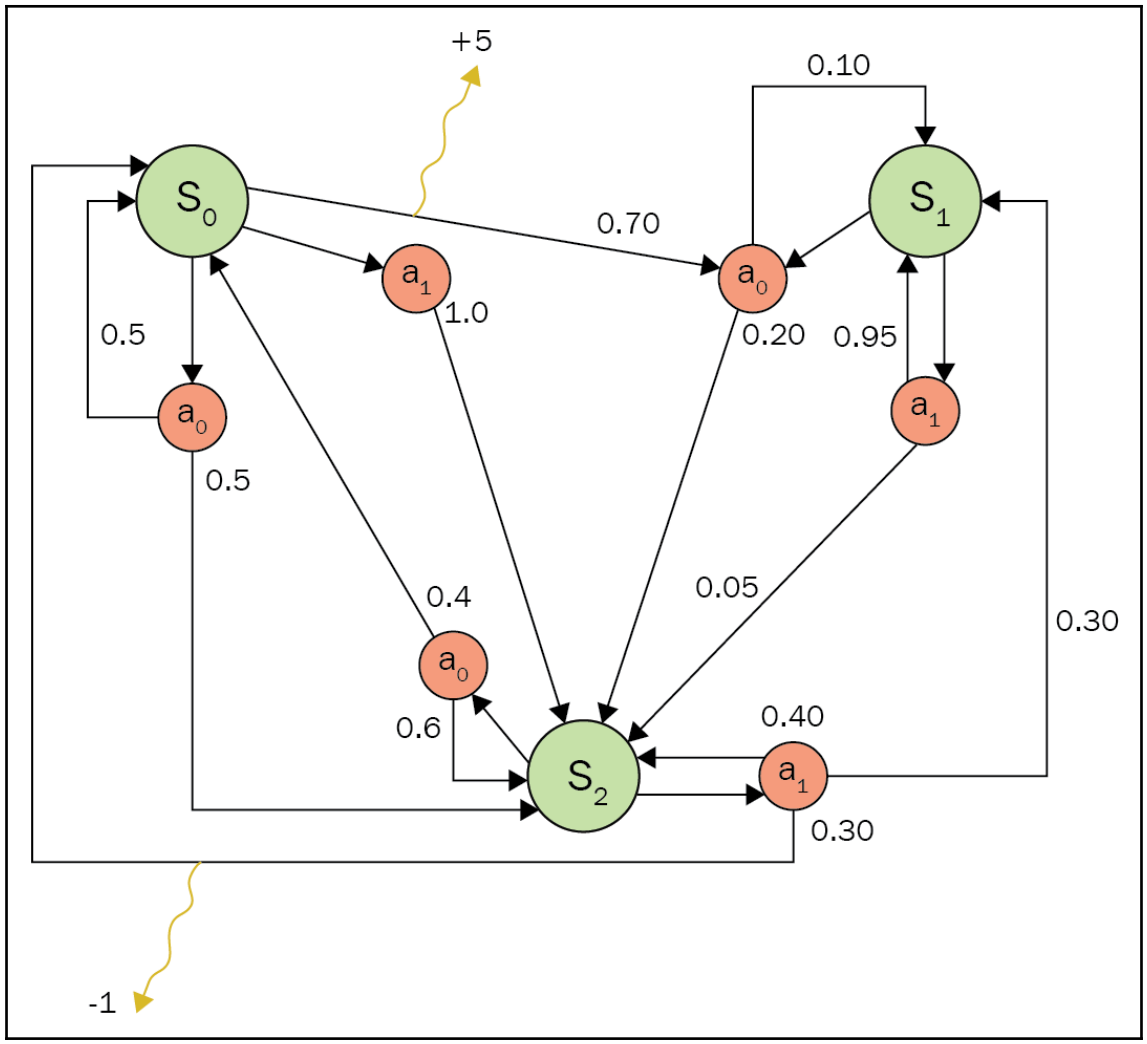


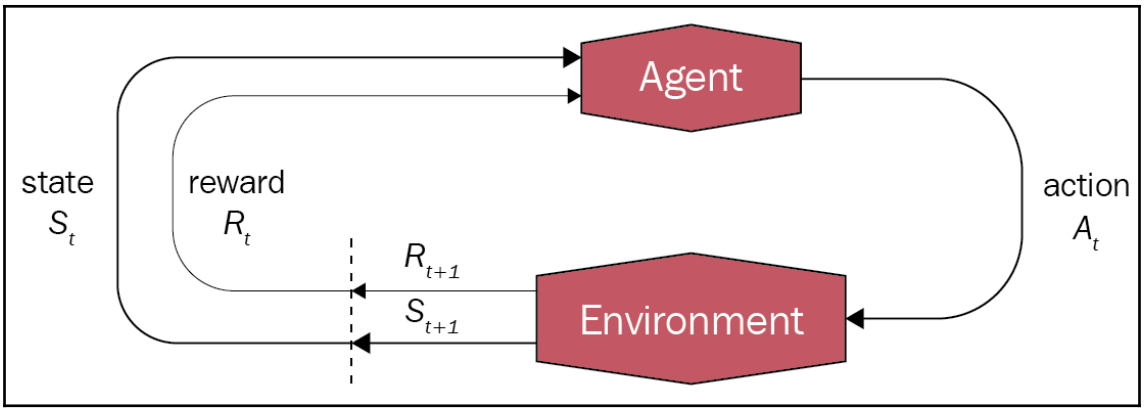



$6
.5
$5
.5
$4


1/4
1/4
1/4
1/4
Head, Head
Head, Tail
Tail, Head
Tail, Tail
H
T
H
T
1/2
1/2
1/2
1/2
H
T
2ndToss
1stToss
1/2
1/2


.5
.5
F
NF
.5
.5

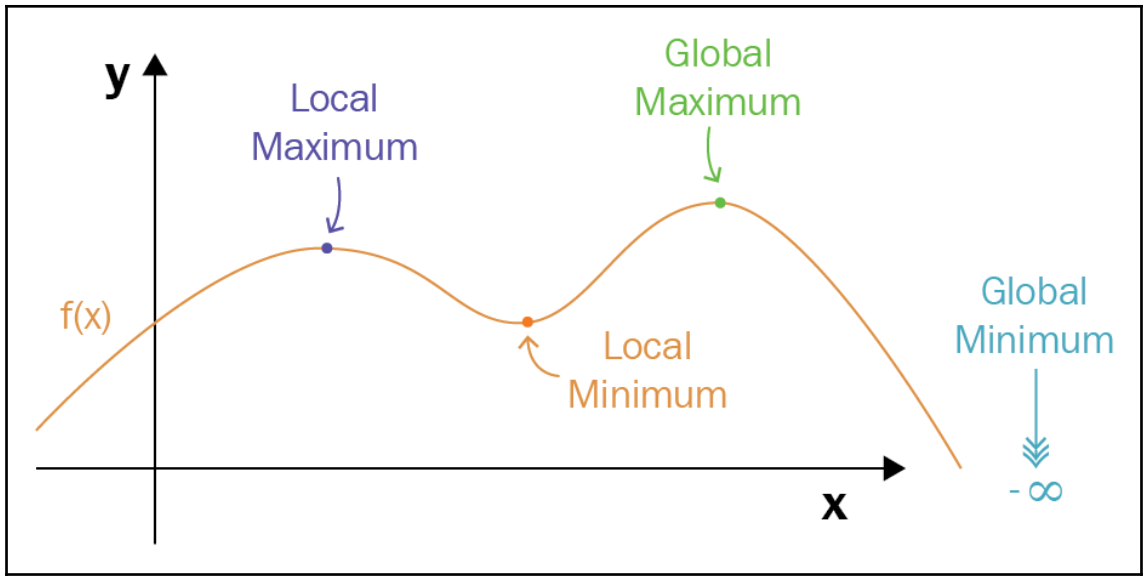

+5
0.10
S0
S1
0.70
a1
a0
1.0
0.20
0.95
0.5
a0
a1
0.5
0.4
0.05
0.30
a0
0.6
0.40
S2
a1
0.30
-1

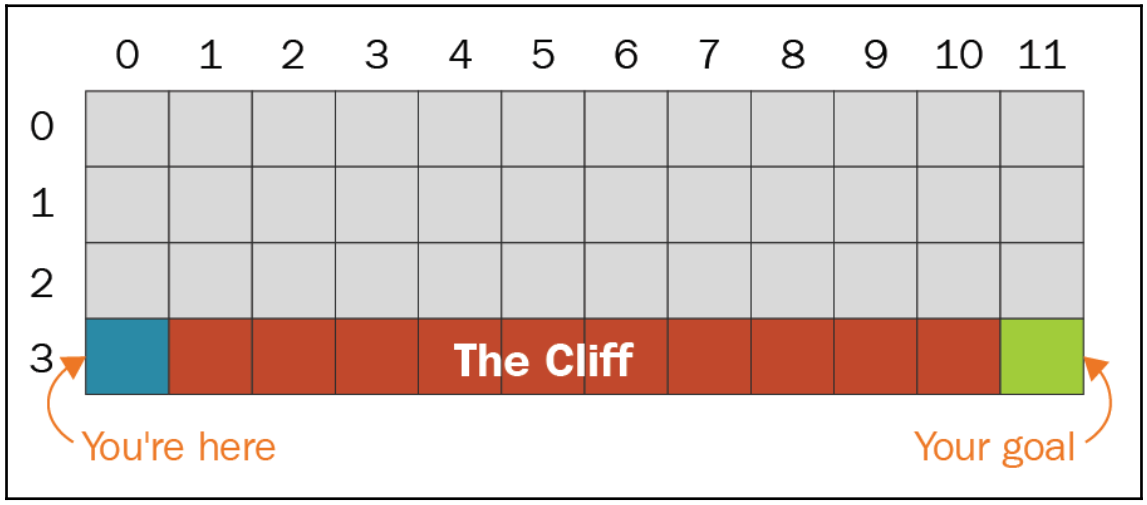



```
import gym
env = gym.make("Taxi-v2")
env.render()
```

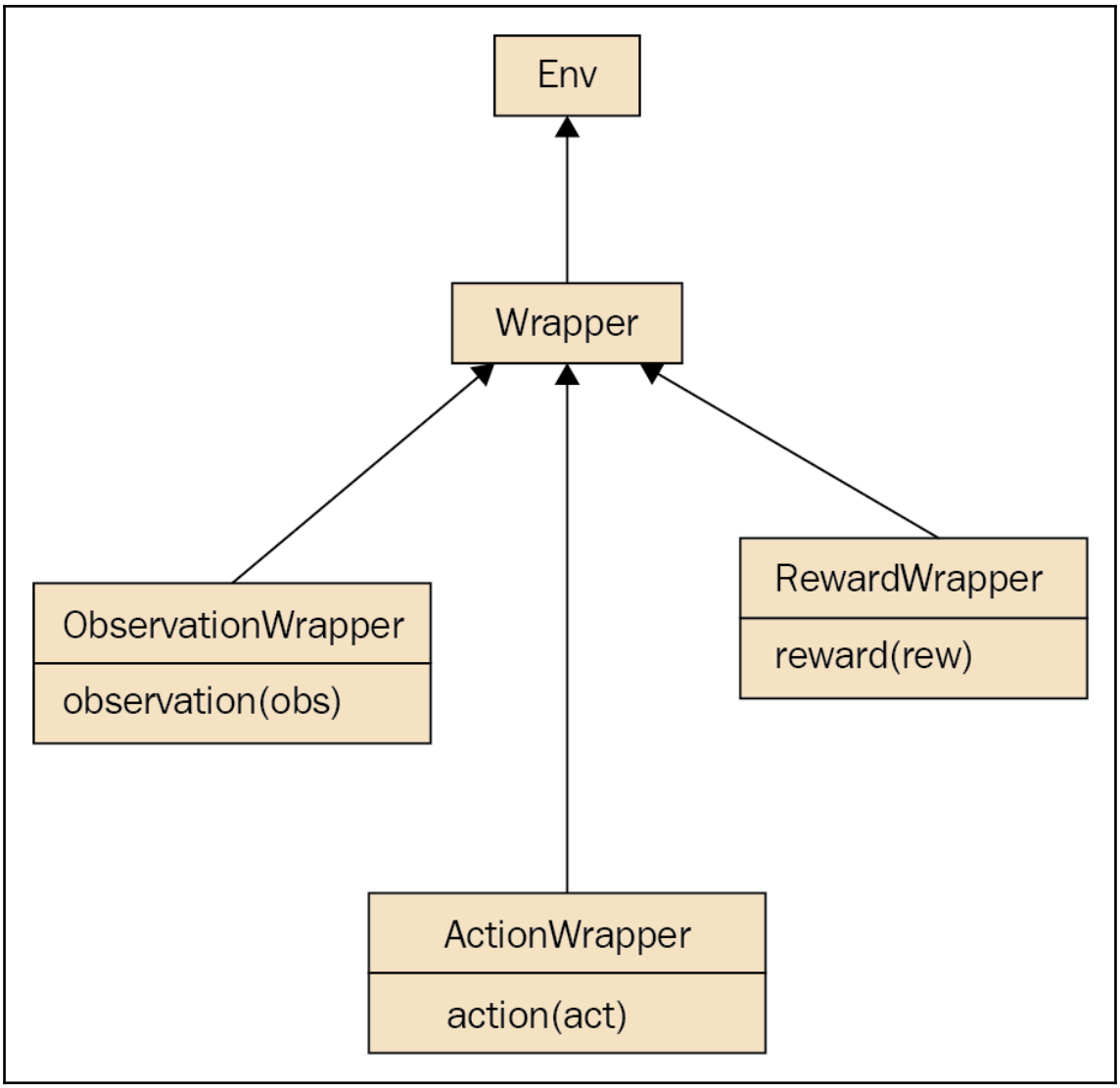
```
+---------+
|R: | : :G|
| : : : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
```

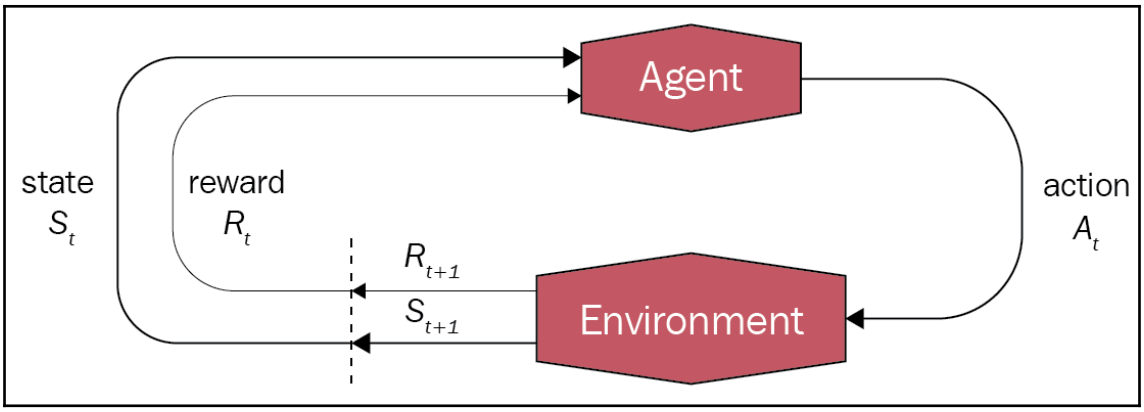
Arm	Reward
1	0
2	1
3	0
4	1
2	0
4	1
4	1
2	0
1	1

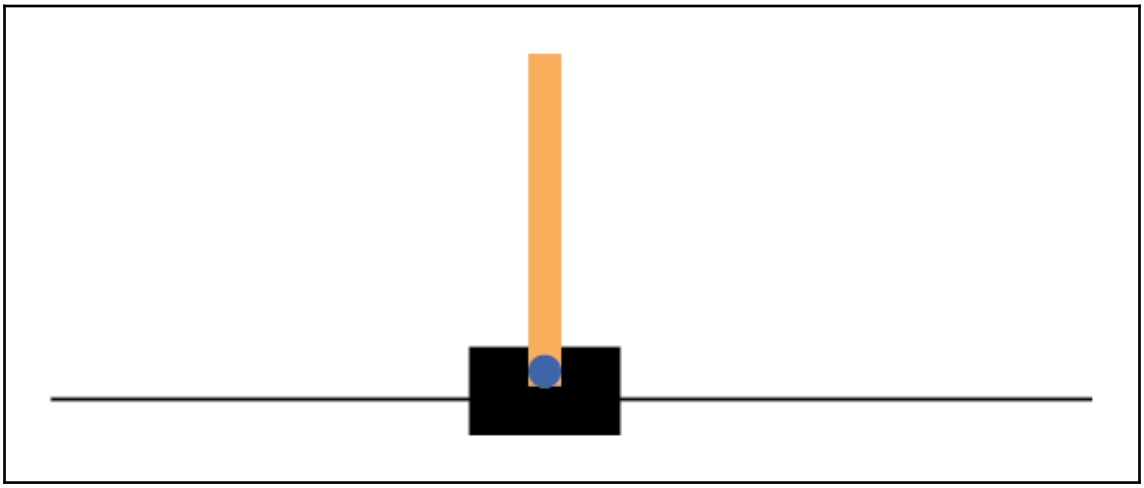


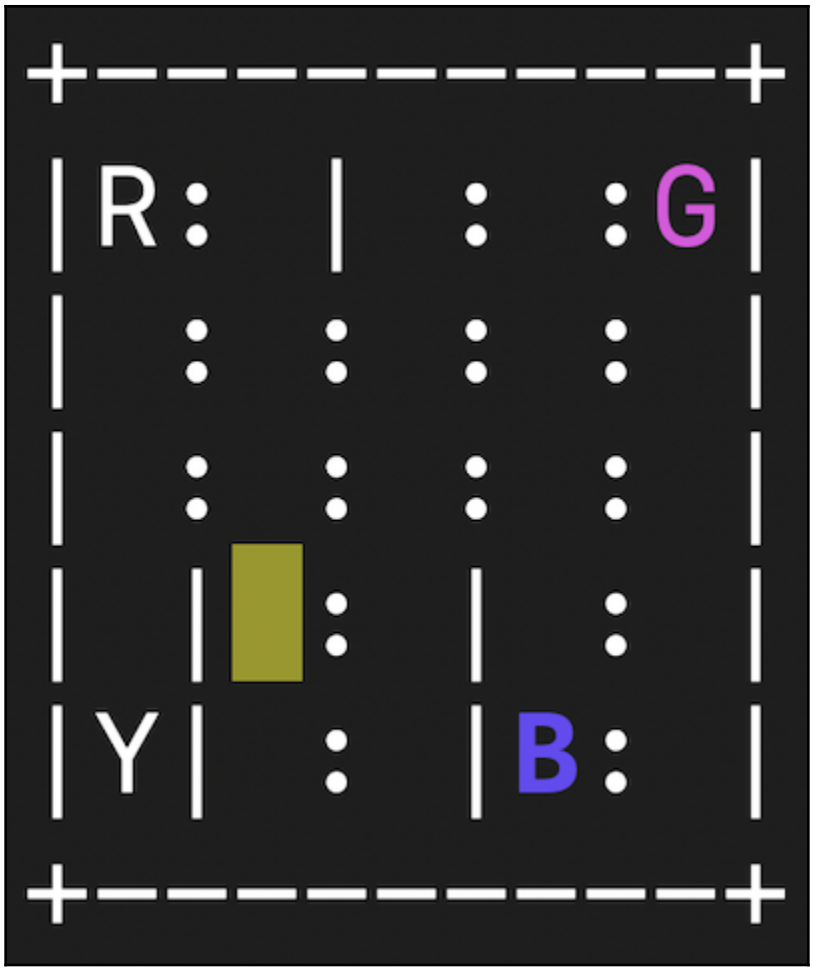


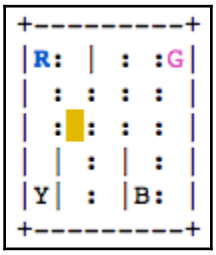
Chapter 3, Setting Up Your First Environment with OpenAI Gym

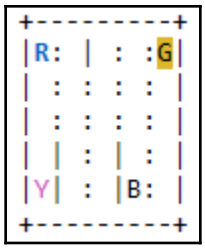

Env
Wrapper
ObservationWrapper
observation(obs)
RewardWrapper
reward(rew)
ActionWrapper
action(act)

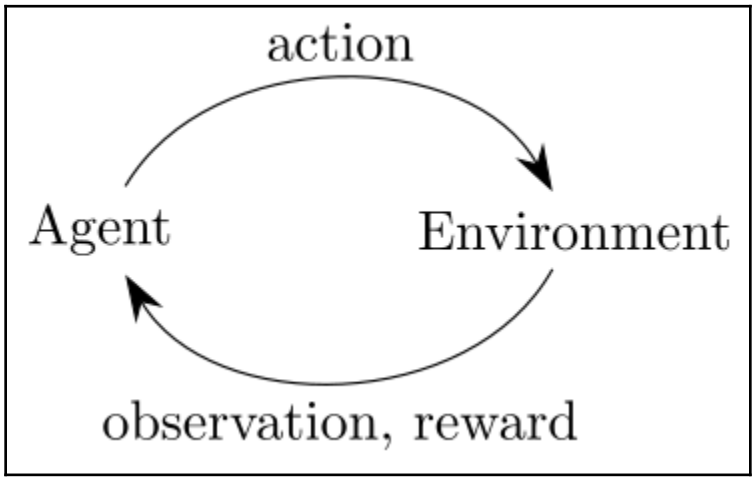

Agent
state
S_t
reward
R_t
R_{t+1}
S_{t+1}
Environment
action
A_t




R
G
Y
B


R
G
Y
B






```
print("Action Space {}".format(env.action_space))
print("State Space {}".format(env.observation_space))
```

```
Action Space Discrete(6)
State Space Discrete(500)
```

```
env.step(1)
```

```
(382, -1, False, {'prob': 1.0})
```

```
env.render()
```

```
+---------+
|R: | : :G|
| : : : : |
| : : : : |
| | : | :▒|
|Y| : |B: |
+---------+
  (North)
```

```
env.env.s = 20
env.render()
```

```
+---------+
|R:▒| : :G|
| : : : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (North)
```

```
env.env.s = 50
env.render()
```

```
+---------+
|R: |▒: :G|
| : : : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (North)
```

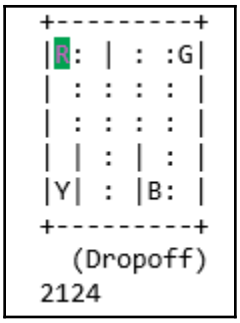
```
env.env.s = 50
env.render()
```

```
+---------+
|R: |: :G|
| : : : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (South)
```

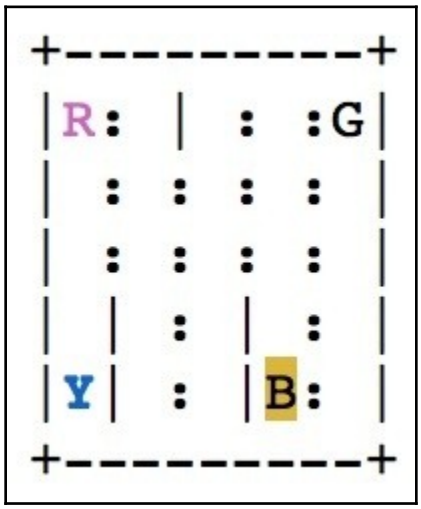
```
env.step(0)
env.render()
```

```
+---------+
|R: | : :G|
| : :: : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (South)
```

```
+---------+
|R: | : :G|
| : : : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (Dropoff)
```


+---------+
|R: | : :G|
| : : : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
(Dropoff)
2124

Chapter 4, Teaching a Smartcab to Drive Using Q-Learning



Q-table initialised at zero

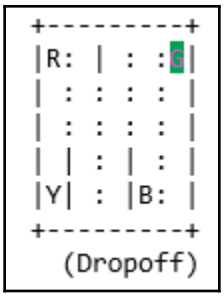
	UP	DOWN	LEFT	RIGHT
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0
7	0	0	0	0
8	0	0	0	0

After few episodes

	UP	DOWN	LEFT	RIGHT
0	0	0	0	0
1	0	0	0	0
2	0	2.25	2.25	0
3	0	0	5	0
4	0	0	0	0
5	0	0	0	0
6	0	5	0	0
7	0	0	2.25	0
8	0	0	0	0

Eventually

	UP	DOWN	LEFT	RIGHT
0	0	0	0.45	0
1	0	1.01	0	0
2	0	2.25	2.25	0
3	0	0	5	0
4	0	0	0	0
5	0	0	0	0
6	0	5	0	0
7	0	0	2.25	0
8	0	0	0	0




```
+---------+
|R: | : :G|
| : : : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (Dropoff)
1998
Counter: 1998
```

```
+---------+
|R: | : :G|
| : : : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (Dropoff)
524
Counter: 524
```

```
total_epochs = 0
episodes = 100

for episode in range(0, episodes):
    epochs = 0
    reward = 0
    state = env.reset()
    while reward != 20:
        action = env.action_space.sample()
        state, reward, done, info = env.step(action)
        epochs += 1
    total_epochs += epochs
print("Average timesteps taken: {}".format(total_epochs/episodes))
```

```
Average timesteps taken: 2464.7
```

```
Q = np.zeros([env.observation_space.n, env.action_space.n])

gamma = 0.1
alpha = 0.1
epsilon = 0.1
total_epochs = 0
episodes = 100

for episode in range(episodes):
    epochs = 0
    reward = 0
    state = env.reset()

    while reward != 20:
        if np.random.rand() < epsilon:
            action = env.action_space.sample()
        else:
            action = np.argmax(Q[state])
        next_state, reward, done, info = env.step(action)
        Q[state, action] = Q[state, action] + alpha * (reward + gamma *   \
                           np.max(Q[next_state]) - Q[state, action])
        state = next_state
        epochs += 1
    total_epochs += epochs

print("Average timesteps taken: {}".format(total_epochs/episodes))
```

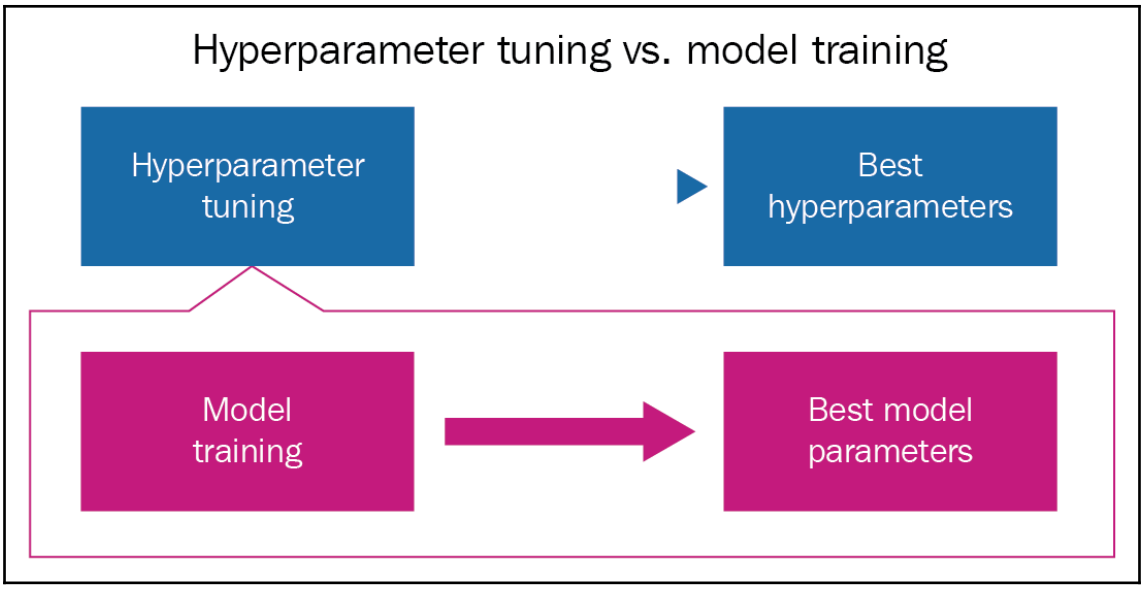
```
Average timesteps taken: 663.97
```

```
total_epochs = 0
episodes = 10000

for episode in range(episodes):
    epochs = 0
    reward = 0
    state = env.reset()
    while reward != 20:
        action = env.action_space.sample()
        state, reward, done, info = env.step(action)
        epochs += 1
    total_epochs += epochs
print("Average timesteps taken: {}".format(total_epochs/episodes))
```

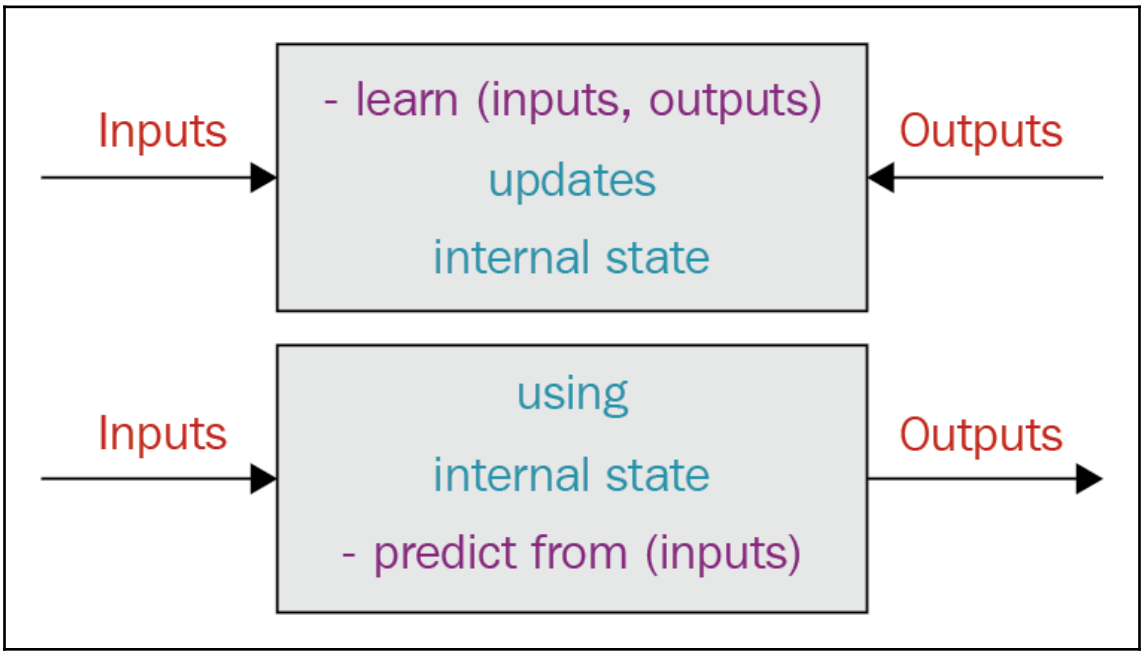
```
Average timesteps taken: 2255.7757
```

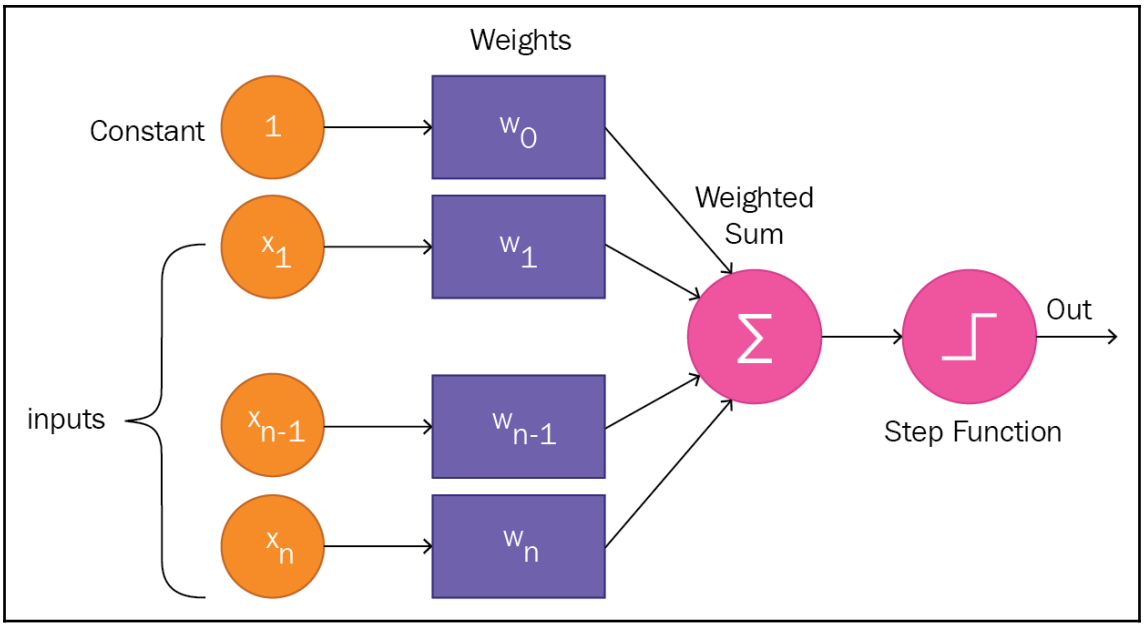
```
Q = np.zeros([env.observation_space.n, env.action_space.n])

gamma = 0.1
alpha = 0.1
epsilon = 0.1
total_epochs = 0
episodes = 10000

for episode in range(episodes):
    epochs = 0
    reward = 0
    state = env.reset()

    while reward != 20:
        if np.random.rand() < epsilon:
            action = env.action_space.sample()
        else:
            action = np.argmax(Q[state])
        next_state, reward, done, info = env.step(action)
        Q[state, action] = Q[state, action] + alpha * (reward + gamma *    \
                           np.max(Q[next_state]) - Q[state, action])
        state = next_state
        epochs += 1
    total_epochs += epochs

print("Average timesteps taken: {}".format(total_epochs/episodes))
```

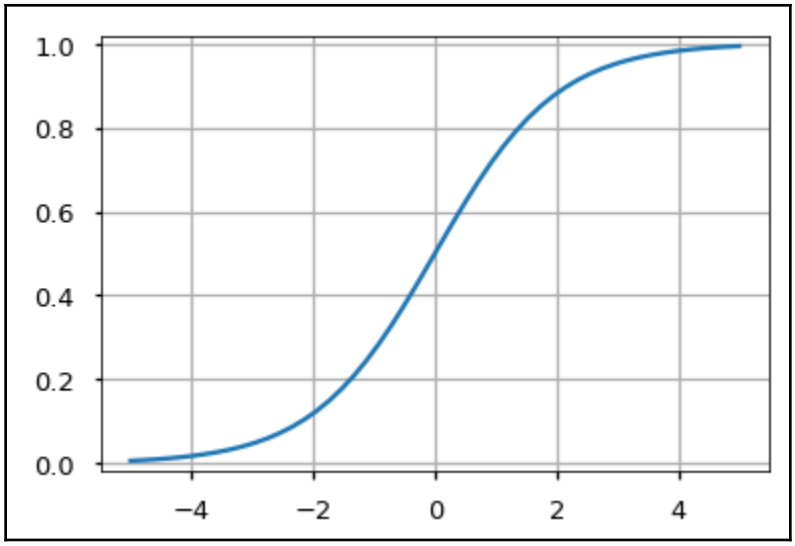
```
Average timesteps taken: 46.0174
```

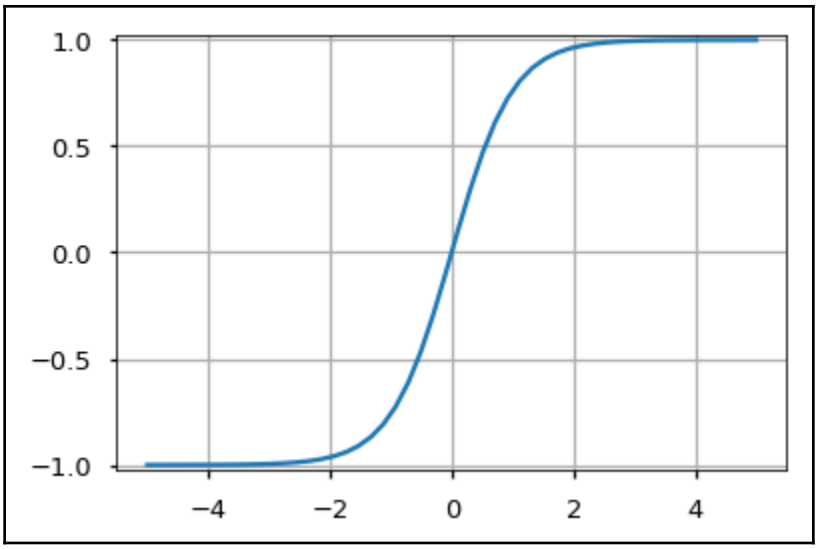

Hyperparameter tuning vs. model training
Hyperparameter tuning
Best hyperparameters
Model training
Best model parameters

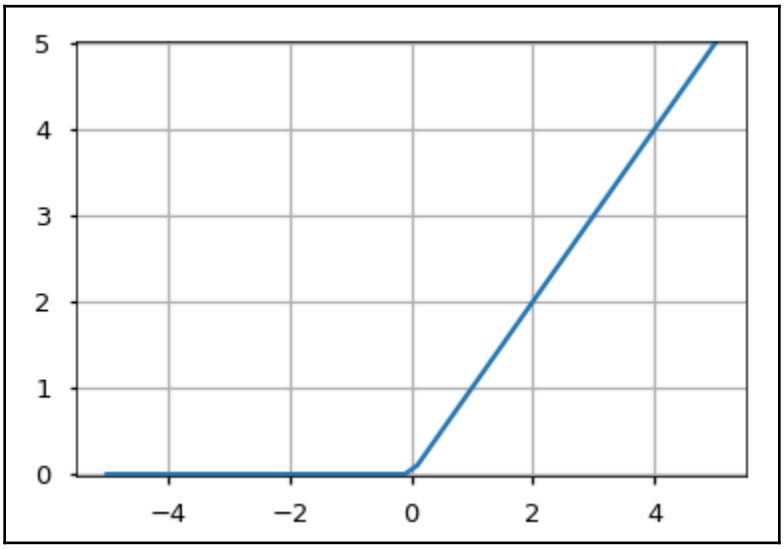
Chapter 5, Building Q-Networks with TensorFlow

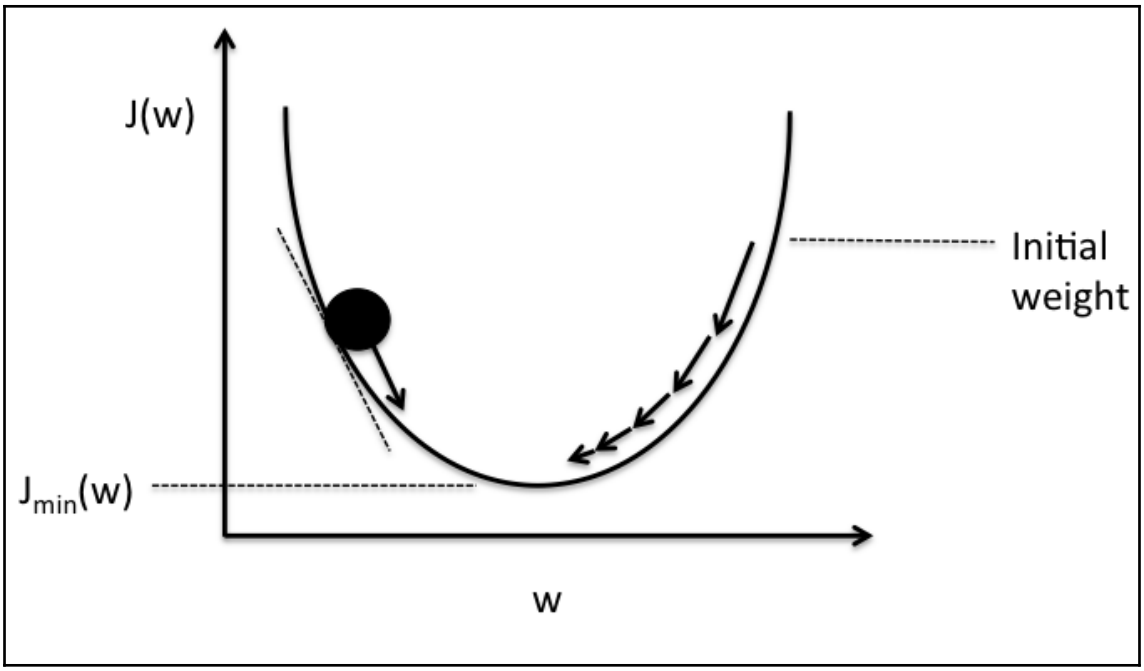


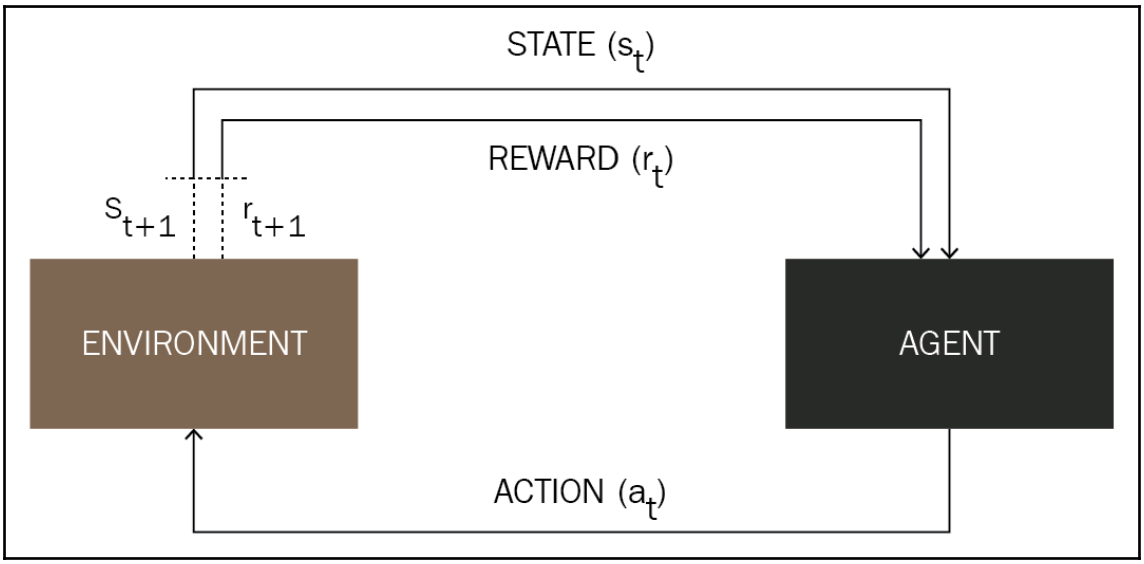

Weights
Constant
1
w_0
x_1
w_1
Weighted
Sum
Out
inputs
x_{n-1}
w_{n-1}
x_n
w_n
Step Function


1.0
0.8
0.6
0.4
0.2
0.0
−4
−2
0
2
4

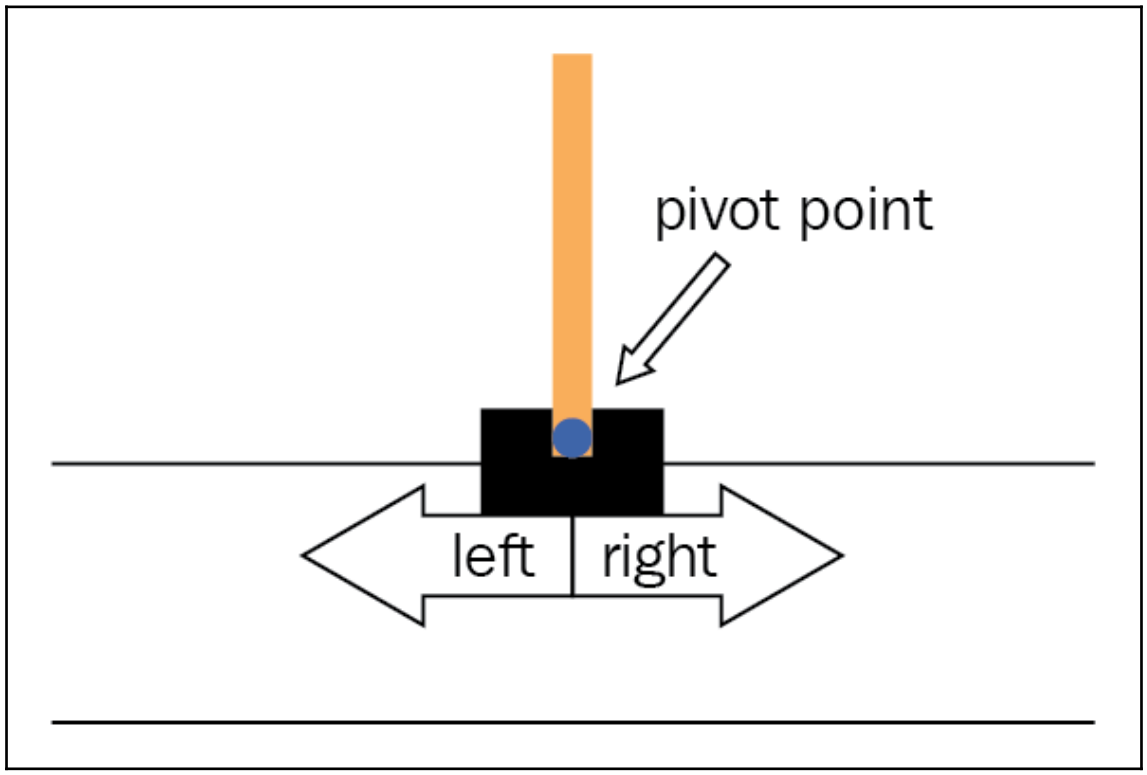

1.0
0.5
0.0
−0.5
−1.0
−4
−2
0
2
4

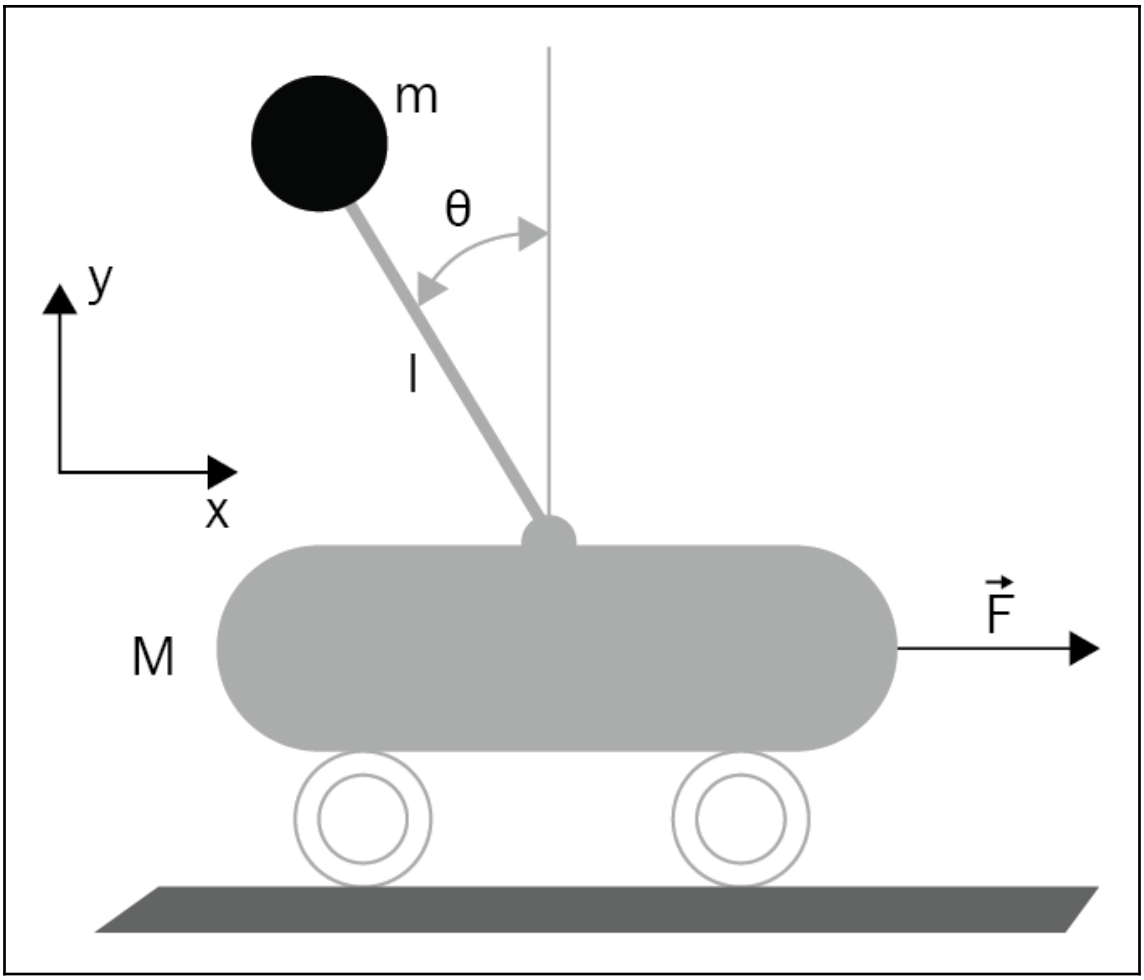

5
4
3
2
1
0
−4
−2
0
2
4

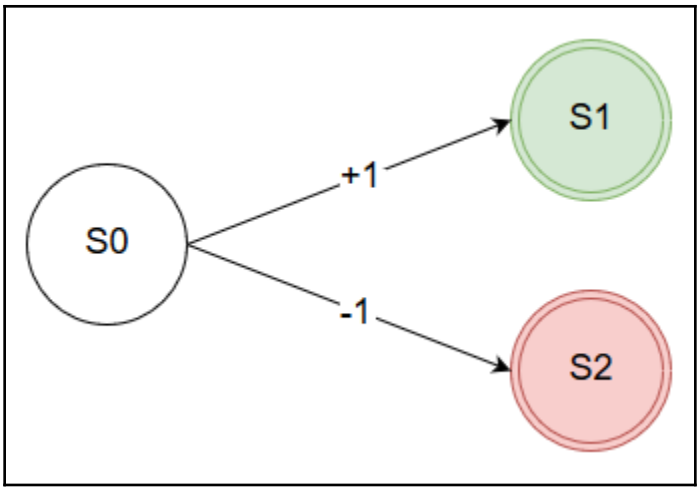

J(w)
Initial
weight
$J_{min}(w)$
w

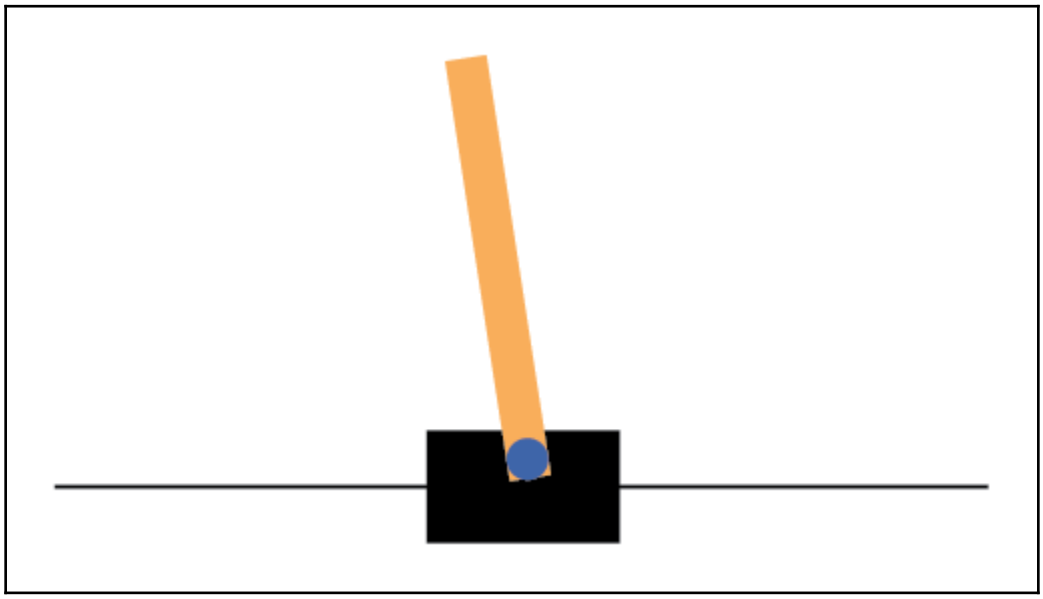

STATE (s_t)
REWARD (r_t)
S_{t+1}
r_{t+1}
ENVIRONMENT
AGENT
ACTION (a_t)

Chapter 6, Digging Deeper into Deep Q-Networks with Keras and TensorFlow




m
θ
y
l
x
M
F


S1
+1
S0
-1
S2

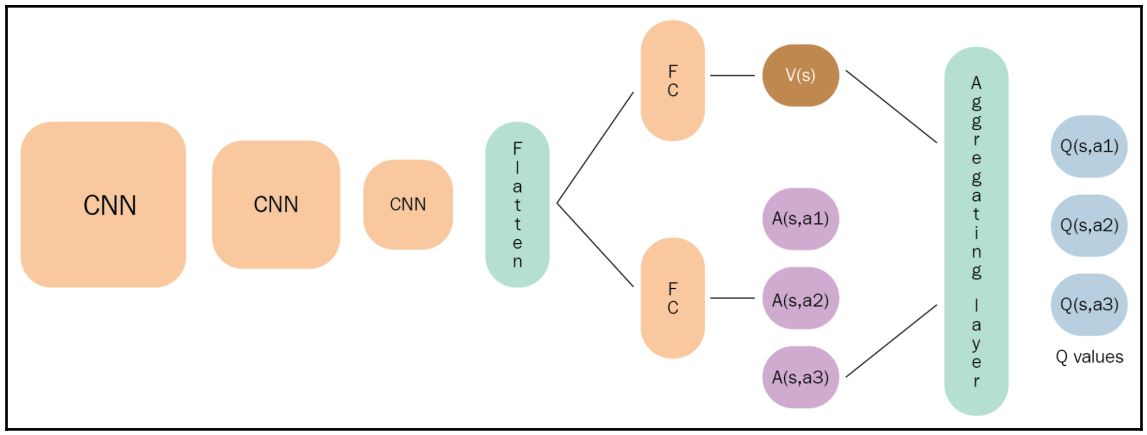


```
if __name__ == "__main__":
    cartpole()
```

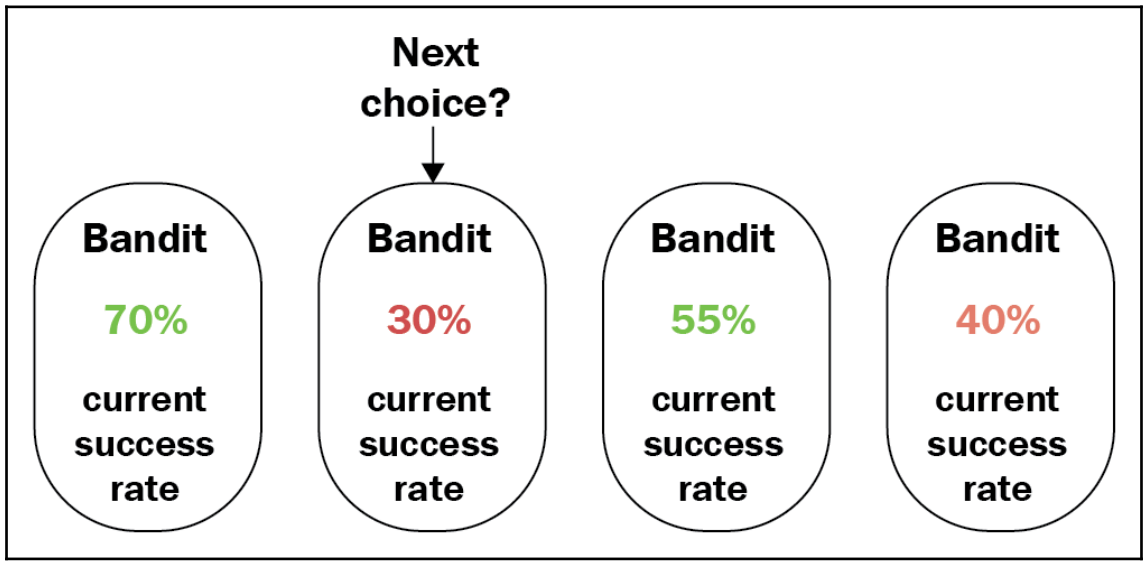
```
Epoch: 450 Score: 12
Epoch: 451 Score: 24
Epoch: 452 Score: 19
Epoch: 453 Score: 15
Epoch: 454 Score: 16
Epoch: 455 Score: 19
Epoch: 456 Score: 38
Epoch: 457 Score: 18
Epoch: 458 Score: 22
Epoch: 459 Score: 13
Epoch: 460 Score: 19
Epoch: 461 Score: 16
Epoch: 462 Score: 28
Epoch: 463 Score: 10
Epoch: 464 Score: 12
Epoch: 465 Score: 21
Epoch: 466 Score: 14
Epoch: 467 Score: 26
Epoch: 468 Score: 21
Epoch: 469 Score: 22
```

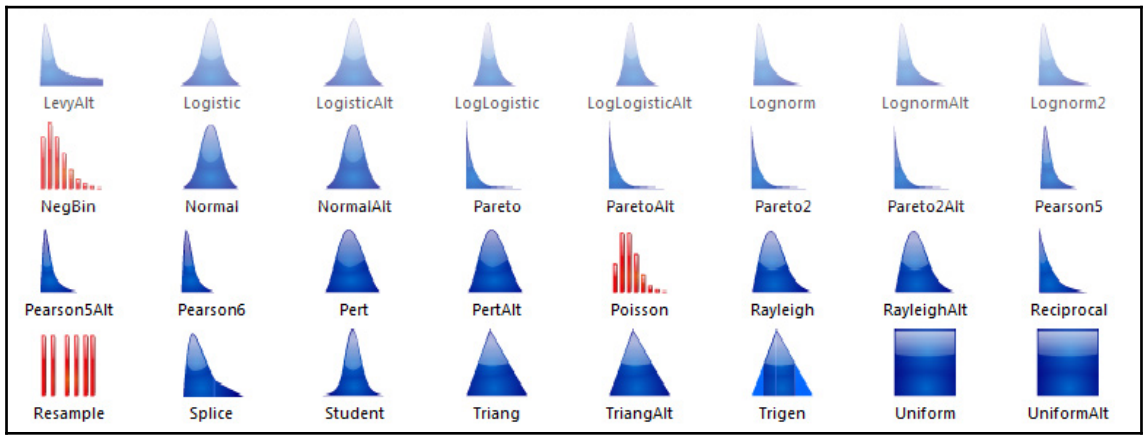
```
if __name__ == "__main__":
    cartpole()
```

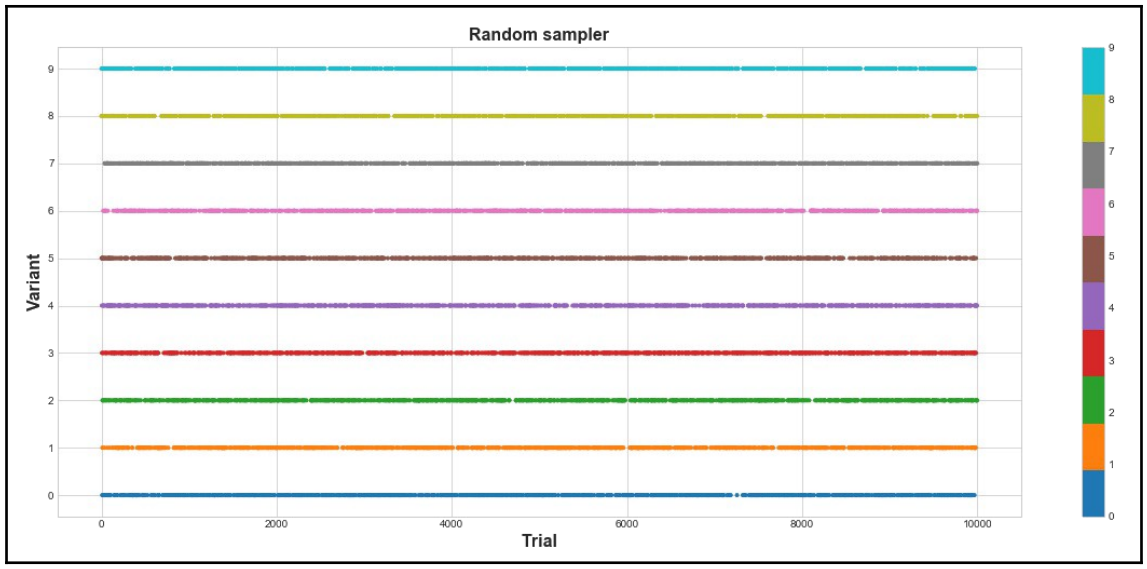
```
Epoch: 1 Score: 30
Epoch: 2 Score: 13
Epoch: 3 Score: 11
Epoch: 4 Score: 10
Epoch: 5 Score: 16
Epoch: 6 Score: 12
Epoch: 7 Score: 13
Epoch: 8 Score: 23
Epoch: 9 Score: 27
Epoch: 10 Score: 9
Epoch: 11 Score: 41
Epoch: 12 Score: 8
Epoch: 13 Score: 12
Epoch: 14 Score: 13
Epoch: 15 Score: 11
Epoch: 16 Score: 24
Epoch: 17 Score: 47
Epoch: 18 Score: 62
Epoch: 19 Score: 62
Epoch: 20 Score: 100
Epoch: 21 Score: 79
Epoch: 22 Score: 124
Epoch: 23 Score: 110
Epoch: 24 Score: 129
Epoch: 25 Score: 126
Epoch: 26 Score: 176
Epoch: 27 Score: 211
Epoch: 28 Score: 201
```

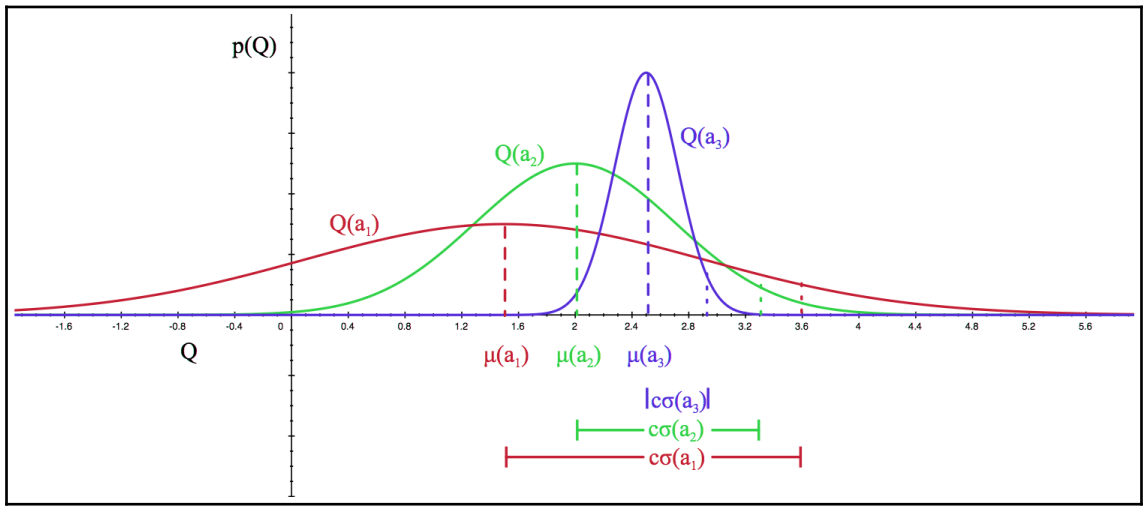

CNN
CNN
CNN
Flatten
FC
FC
V(s)
A(s,a1)
A(s,a2)
A(s,a3)
Aggregating layer
Q(s,a1)
Q(s,a2)
Q(s,a3)
Q values

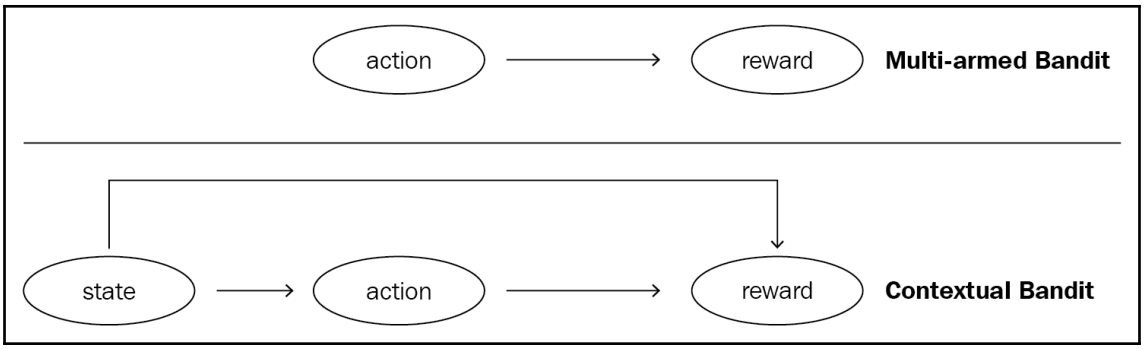
Chapter 7, Decoupling Exploration and Exploitation in Multi-Armed Bandits

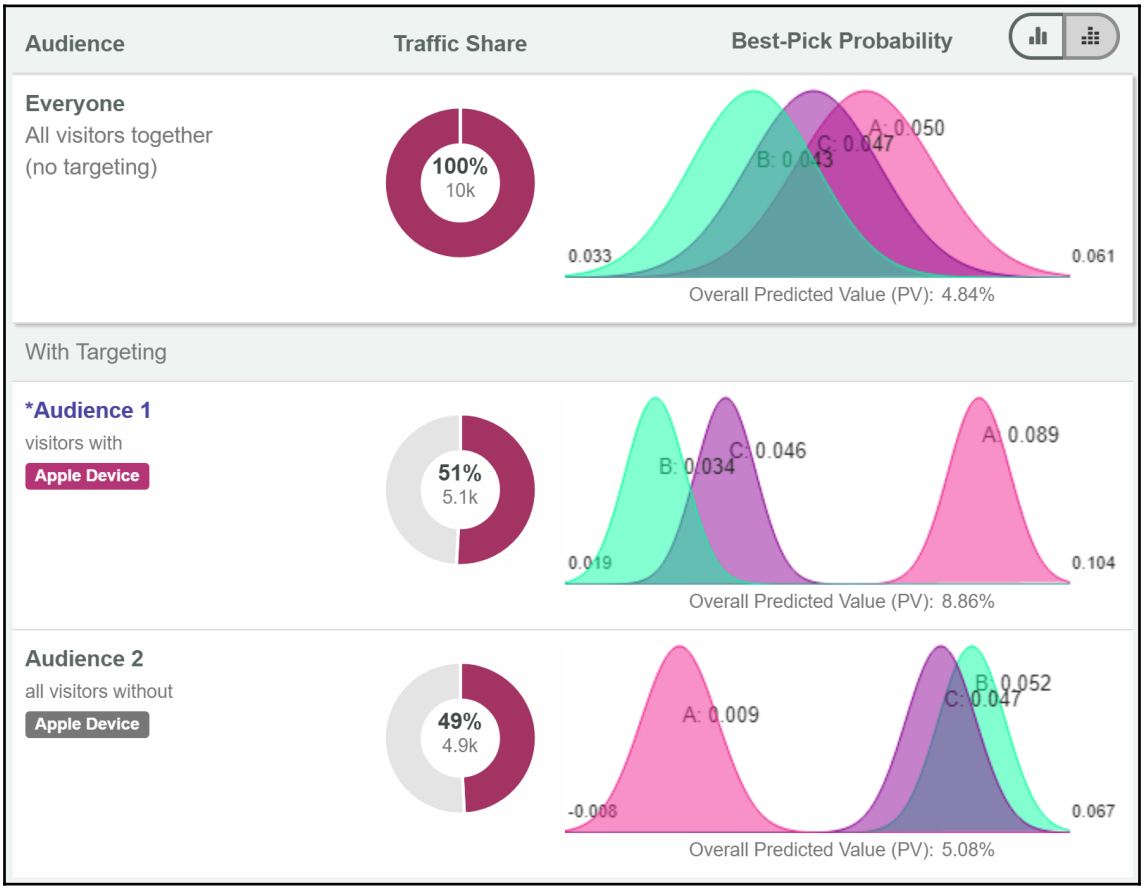


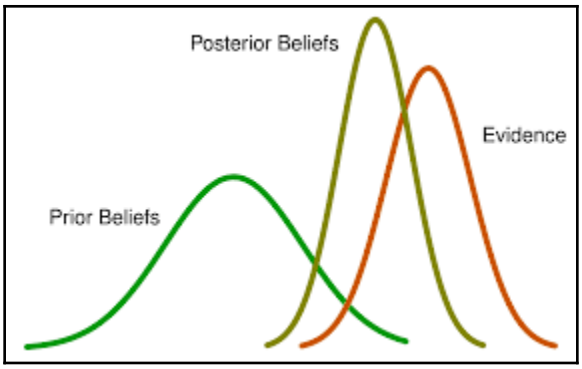



Random sampler
Variant
Trial
0
1
2
3
4
5
6
7
8
9
0
2000
4000
6000
8000
10000


p(Q)
Q
$Q(a_1)$
$Q(a_2)$
$Q(a_3)$
$\mu(a_1)$
$\mu(a_2)$
$\mu(a_3)$
$c\sigma(a_3)$
$c\sigma(a_2)$
$c\sigma(a_1)$
-1.6
-1.2
-0.8
-0.4
0
0.4
0.8
1.2
1.6
2
2.4
2.8
3.2
3.6
4
4.4
4.8
5.2
5.6


action
reward
Multi-armed Bandit
state
action
reward
Contextual Bandit


Audience
Traffic Share
Best-Pick Probability
Everyone
All visitors together
(no targeting)
100%
10k
A: 0.050
B: 0.043
C: 0.047
0.033
0.061
Overall Predicted Value (PV): 4.84%
With Targeting
*Audience 1
visitors with
Apple Device
51%
5.1k
A: 0.089
B: 0.034
C: 0.046
0.019
0.104
Overall Predicted Value (PV): 8.86%
Audience 2
all visitors without
Apple Device
49%
4.9k
A: 0.009
B: 0.052
C: 0.047
-0.008
0.067
Overall Predicted Value (PV): 5.08%

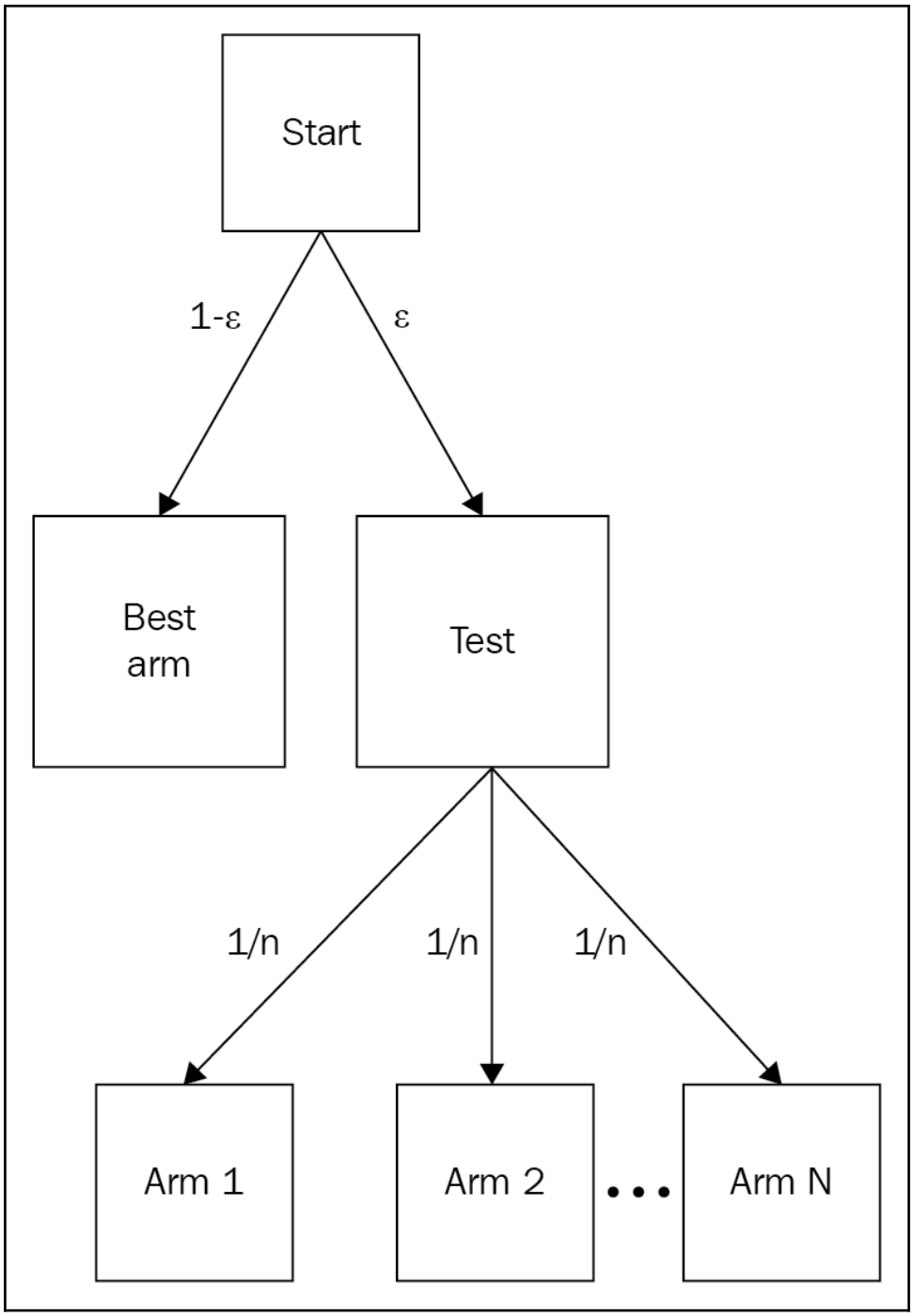


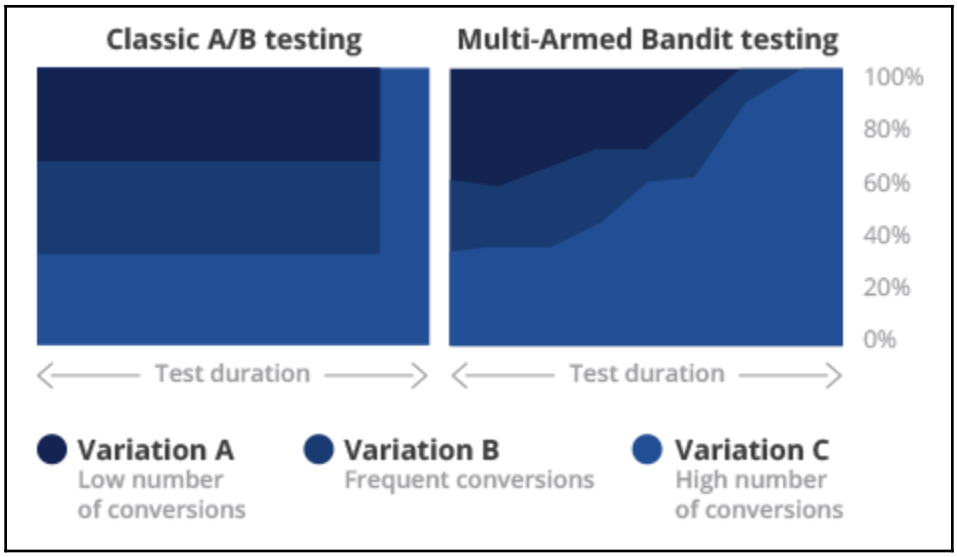
In [28]: `df`

Out[28]:

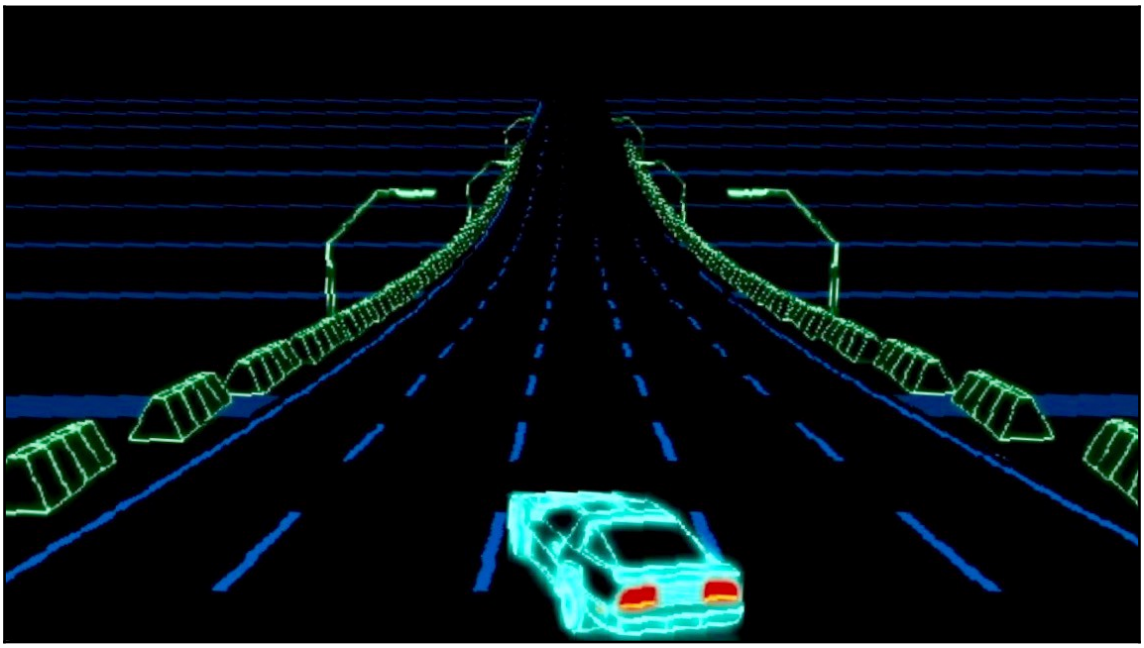
	A	B	C	D	E
0	0	0	0	0	0
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0
5	0	1	0	0	1
6	0	0	0	0	0
7	0	0	0	0	0
8	0	1	0	0	0
9	0	0	0	0	0
10	1	0	0	0	0
11	0	0	0	0	0
12	0	0	0	0	0

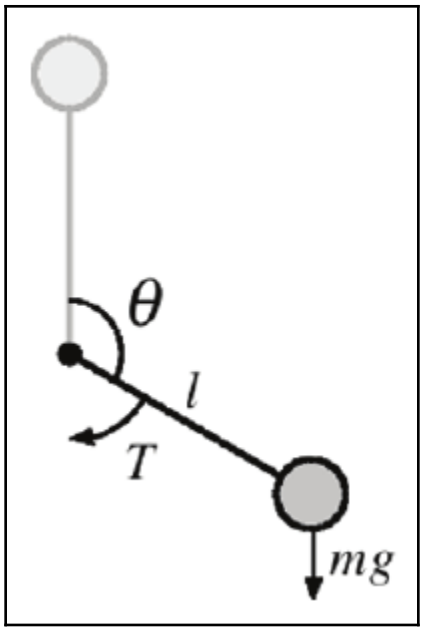
```
In [33]: pd.Series(ad_list).value_counts(normalize=True)

Out[33]: 0    0.2057
         2    0.2006
         3    0.2000
         4    0.1991
         1    0.1946
         dtype: float64
```

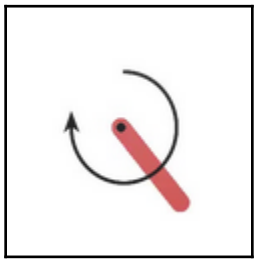

Start
1-ε
ε
Best arm
Test
1/n
1/n
1/n
Arm 1
Arm 2
Arm N

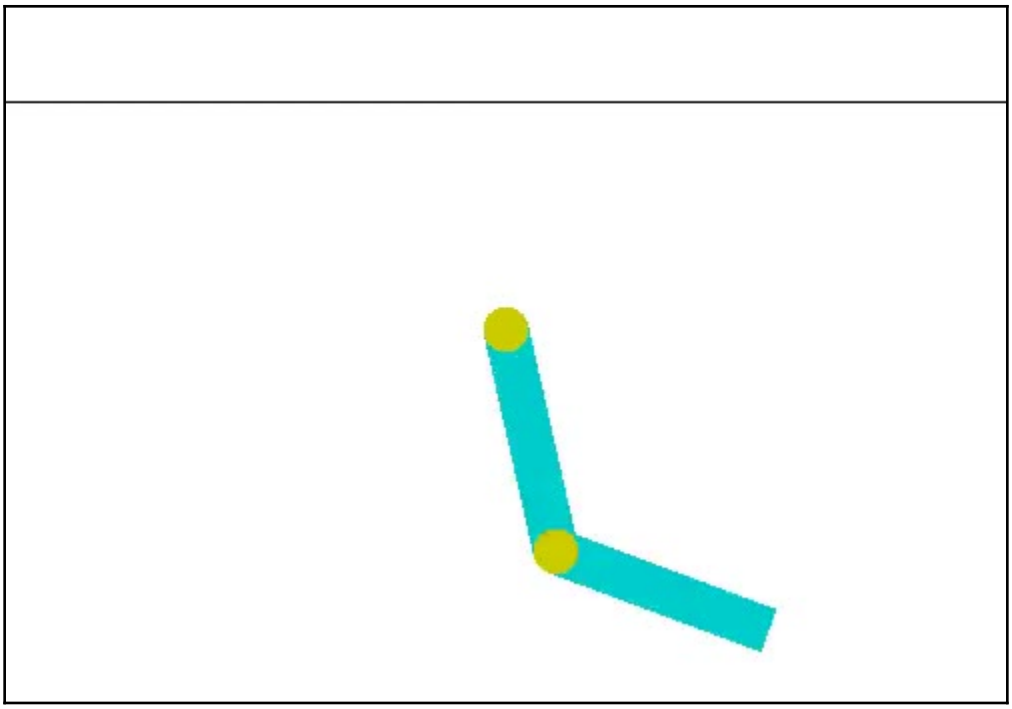

Classic A/B testing
Multi-Armed Bandit testing
100%
80%
60%
40%
20%
0%
Test duration
Test duration
Variation A
Low number of conversions
Variation B
Frequent conversions
Variation C
High number of conversions

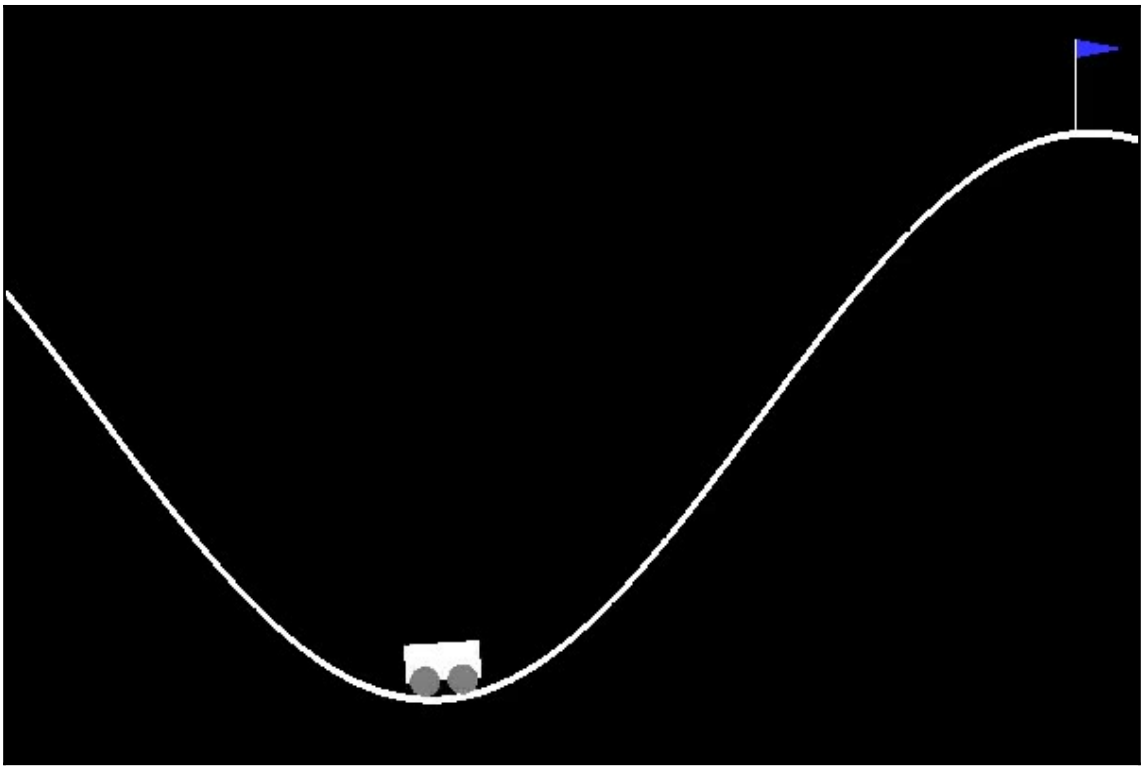
Chapter 8, Further Q-Learning Research and Future Projects

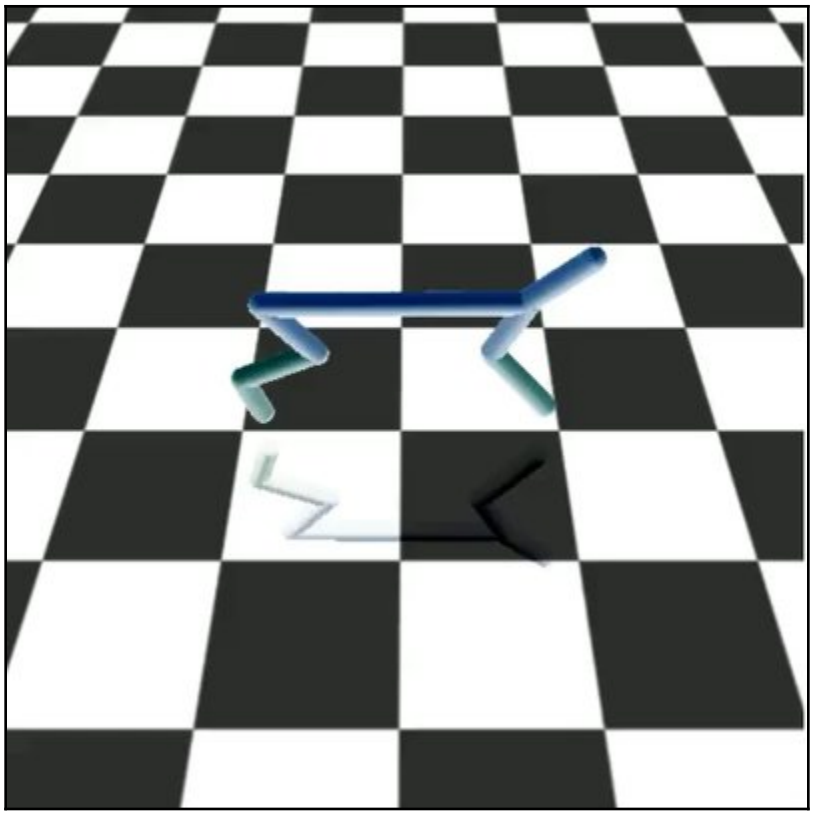


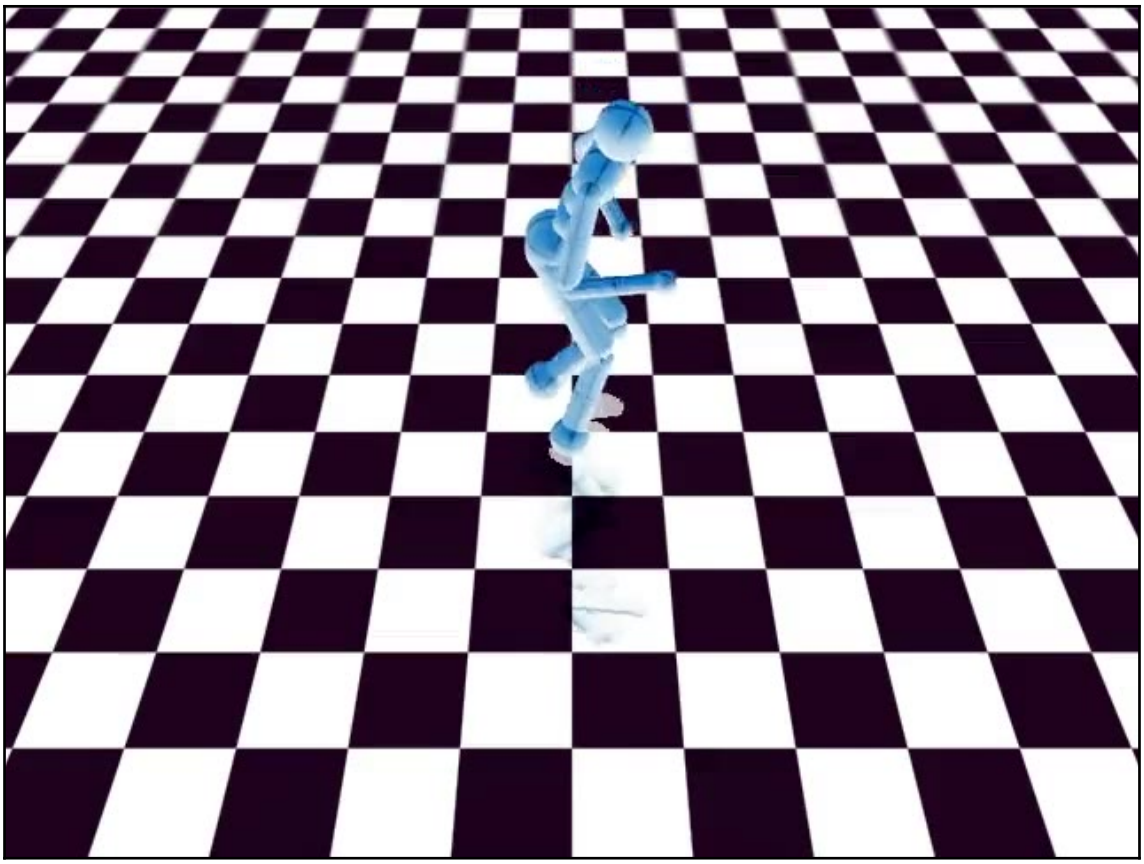

θ
l
T
mg

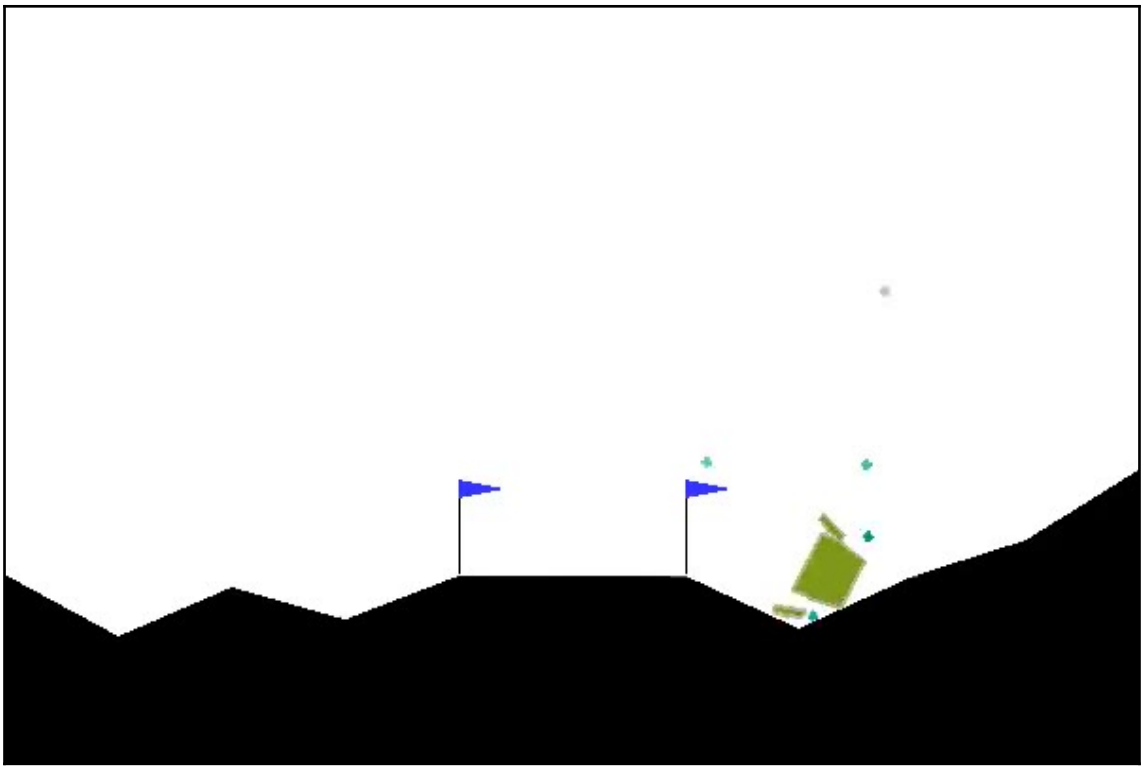


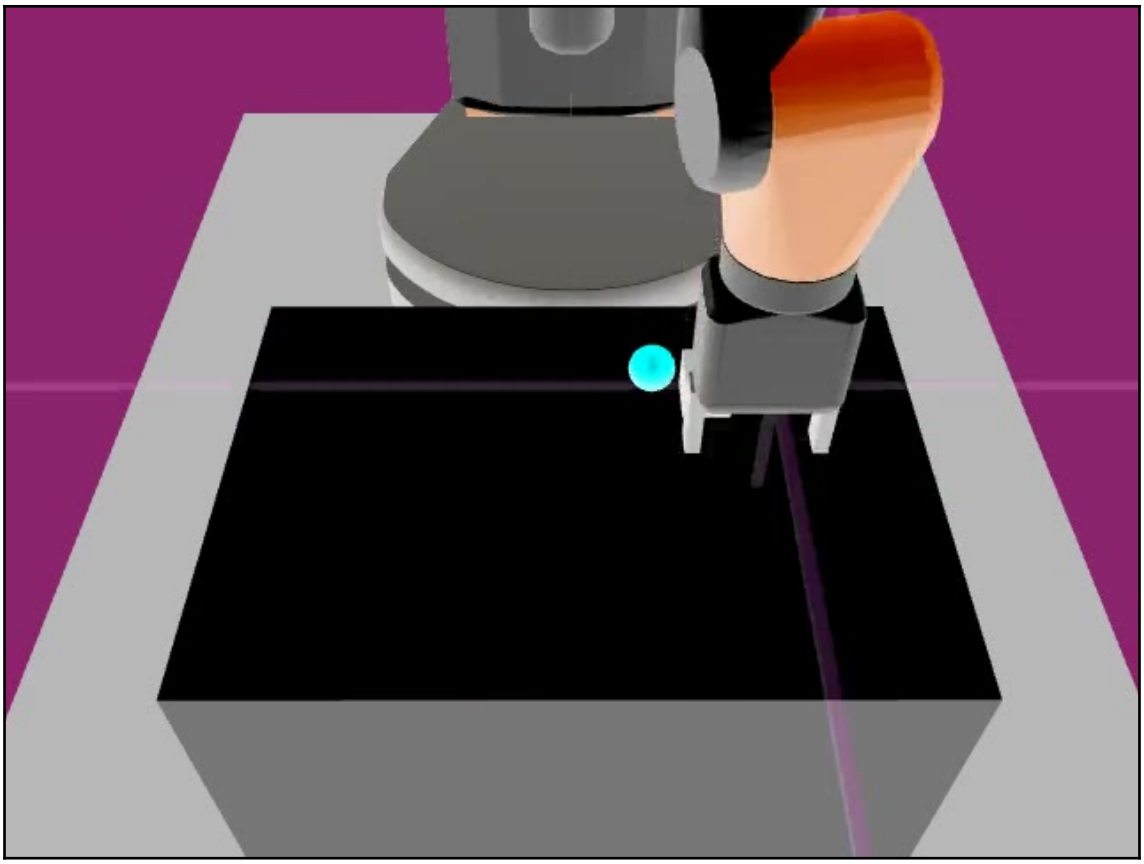




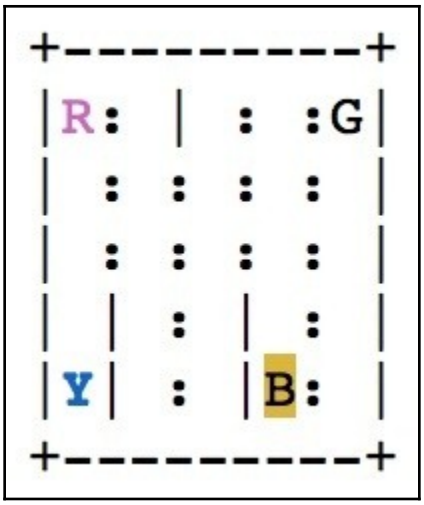


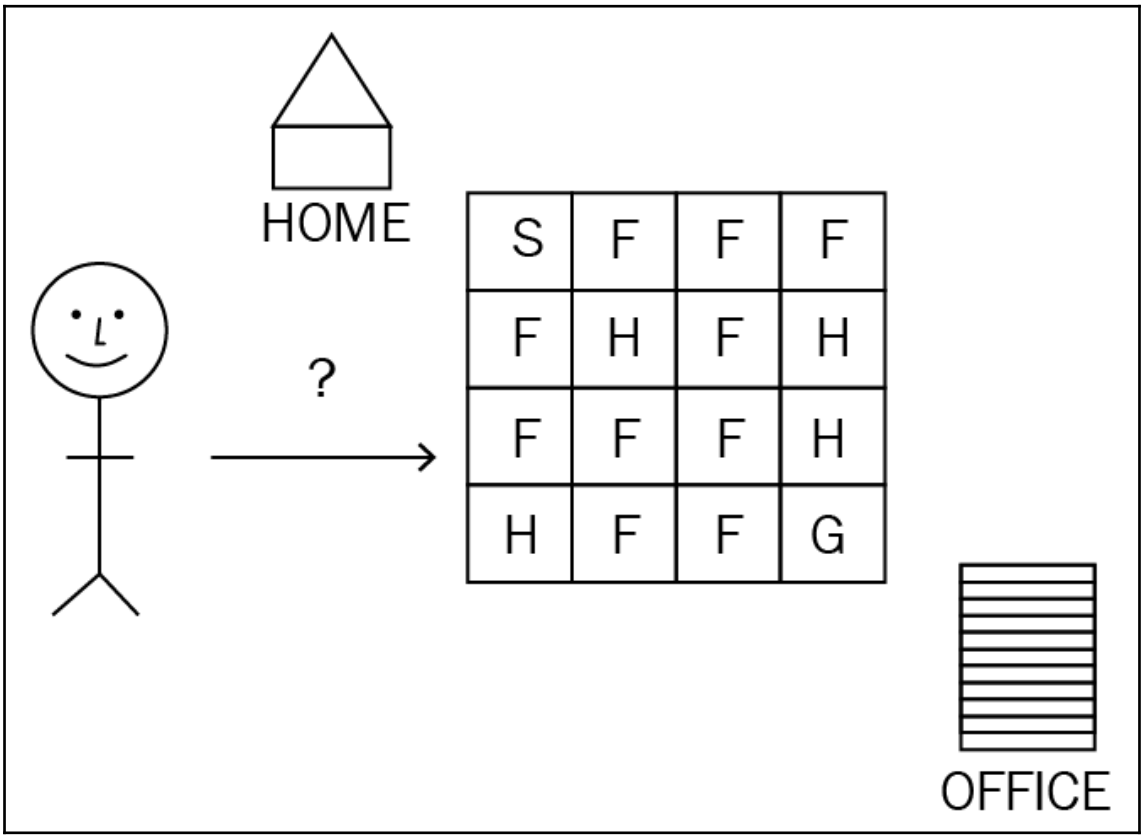


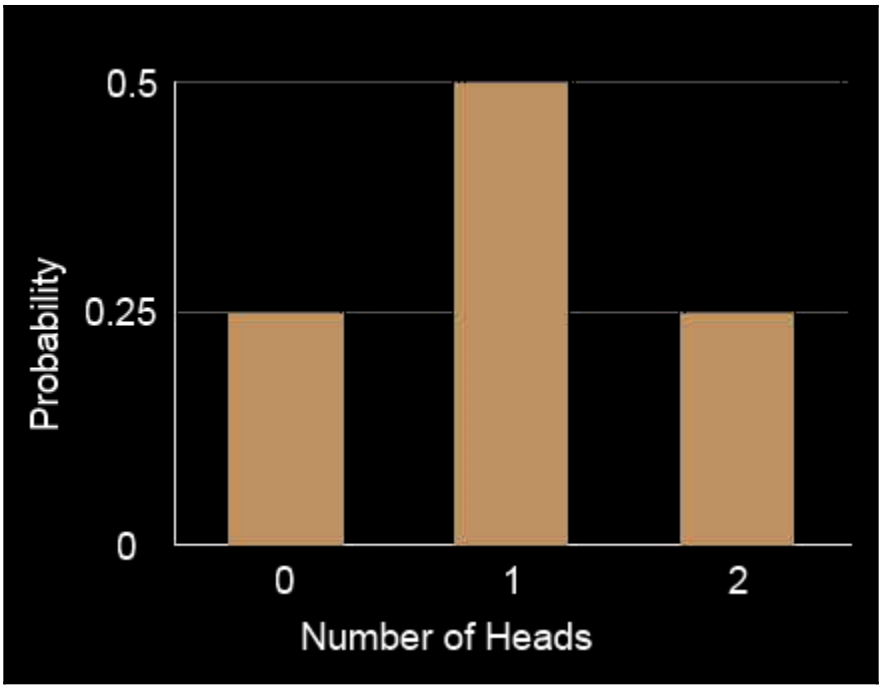


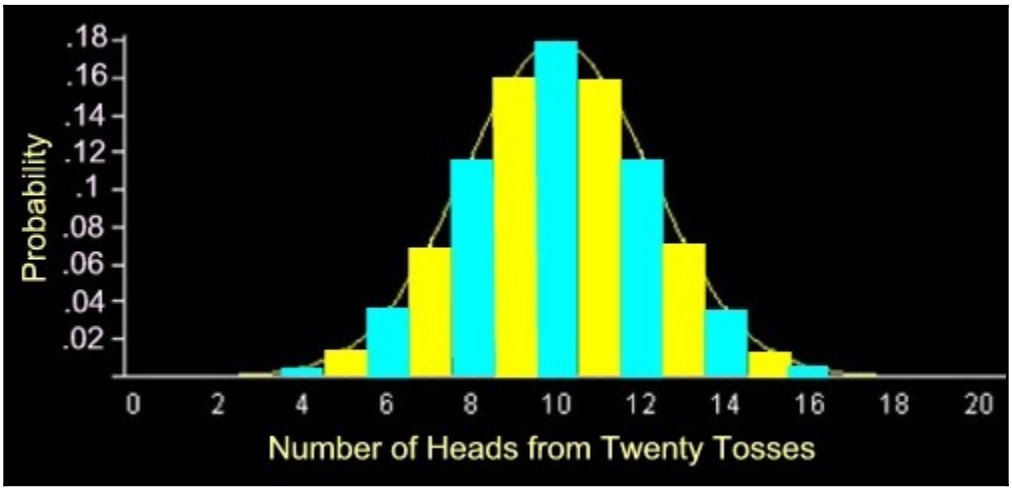


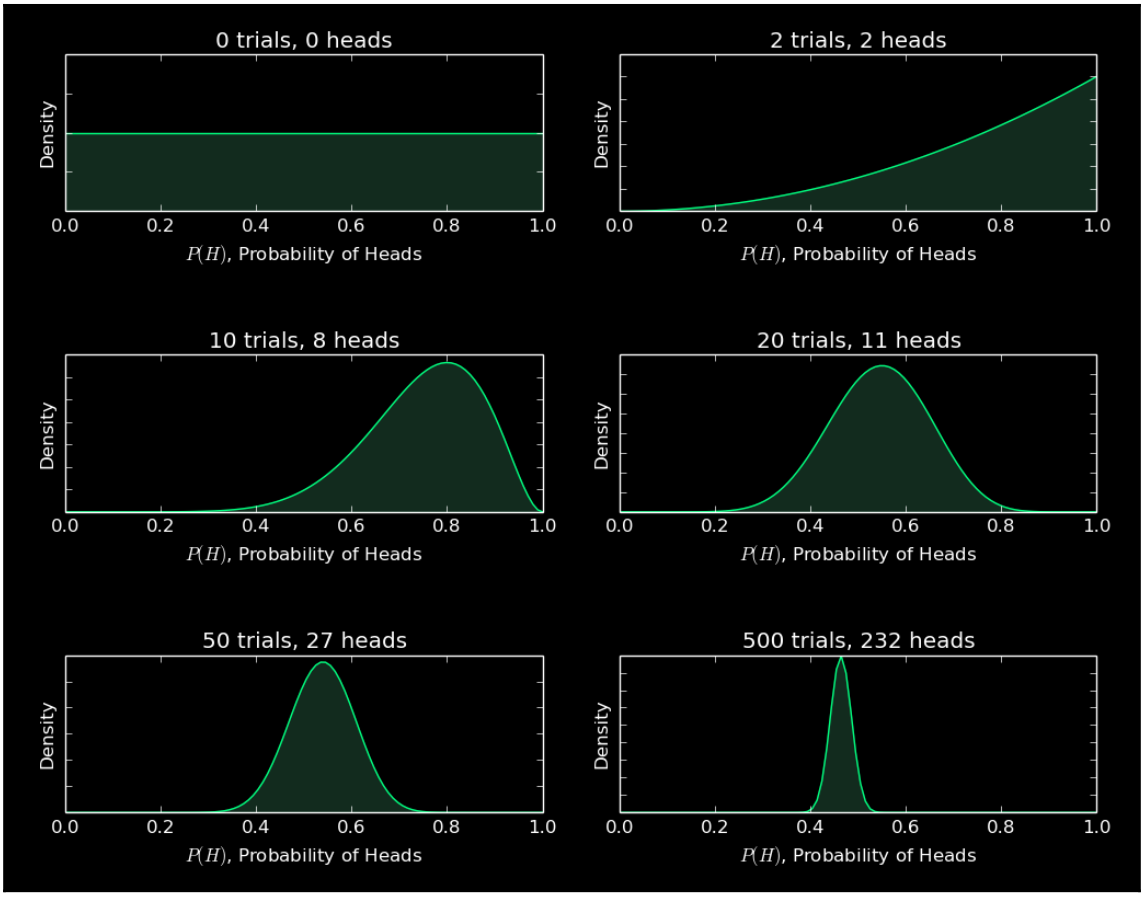

```
Total length of input instance: 4, step: 3
=========================================
Observation Tape    :   DADC
Output Tape         :   DAD
Targets             :   DADC

Current reward      :   1.000
Cumulative reward   :   3.000
Action              :   Tuple(move over input: right,
                              write to the output tape: True
                              prediction: D)
```

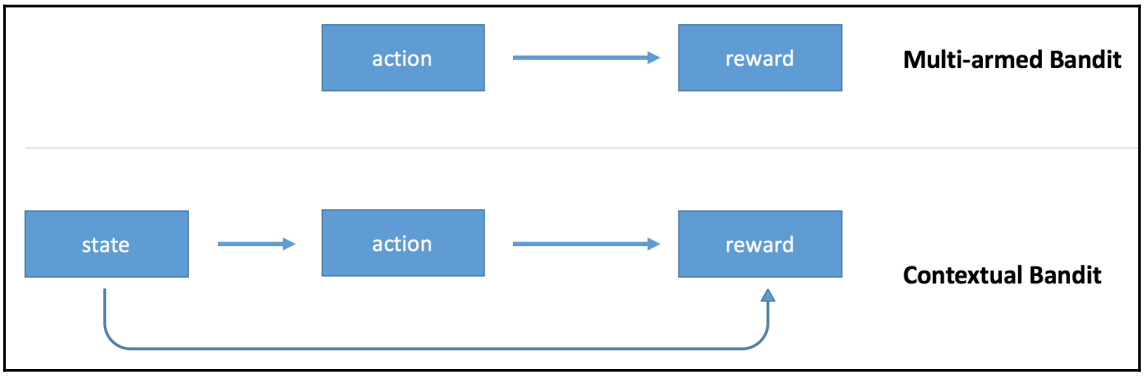

+---------+
|R: | : :G|
| : : : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+

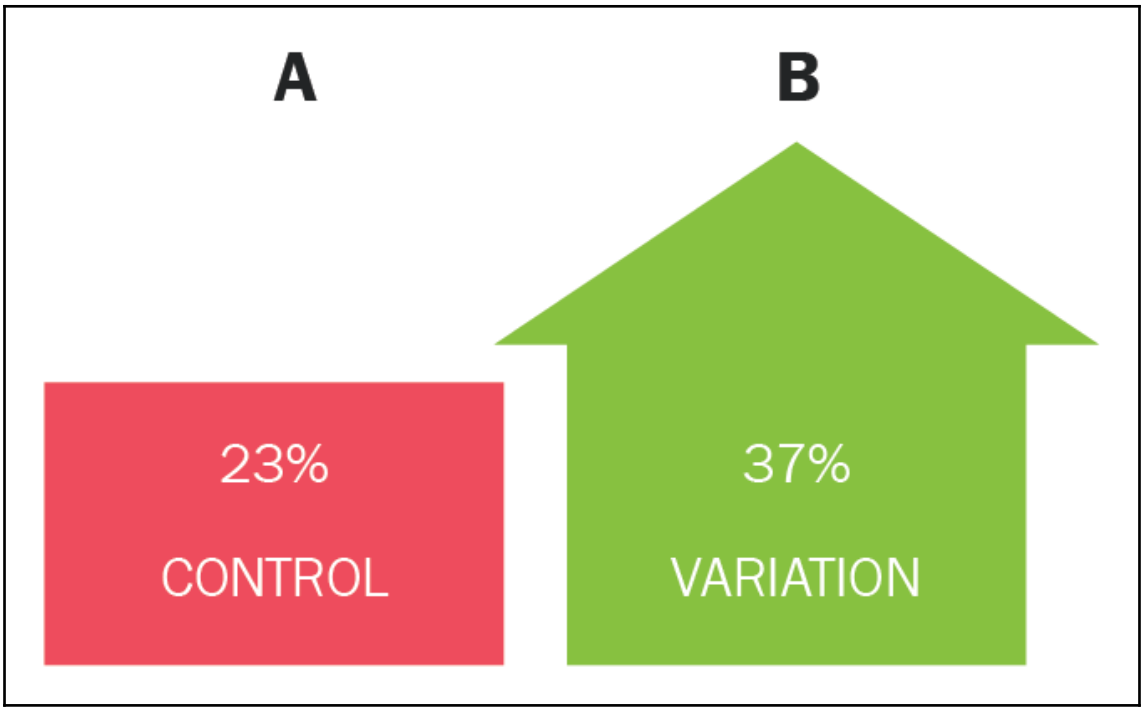

HOME
?
S F F F
F H F H
F F F H
H F F G
OFFICE

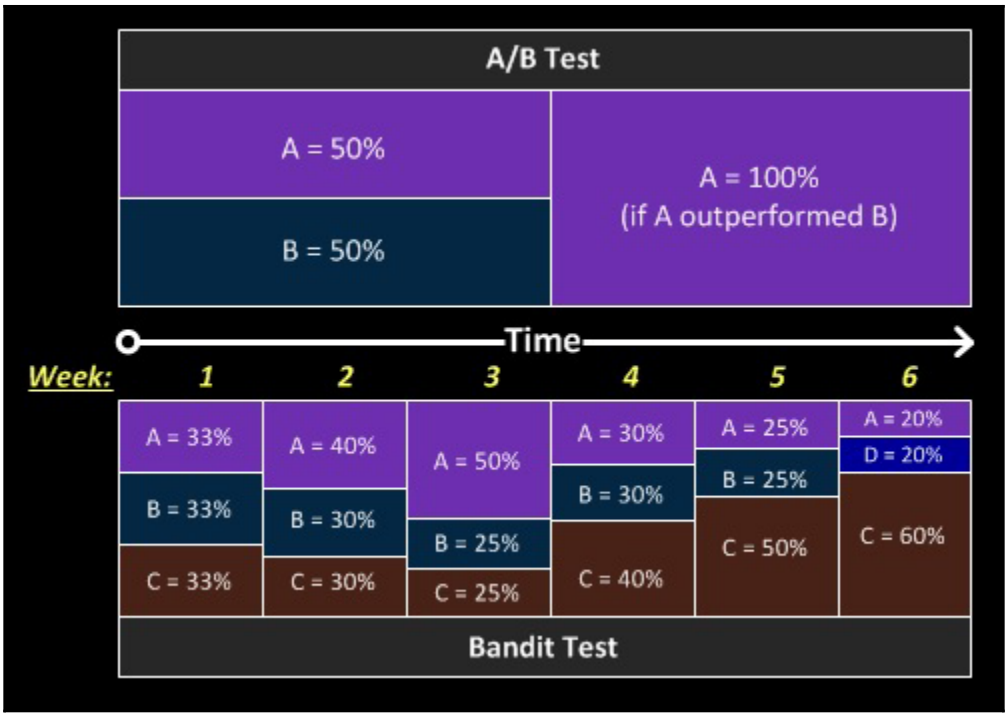

0.5
0.25
0
Probability
0
1
2
Number of Heads


.18
.16
.14
.12
.1
.08
.06
.04
.02
Probability
0
2
4
6
8
10
12
14
16
18
20
Number of Heads from Twenty Tosses


0 trials, 0 heads
2 trials, 2 heads
10 trials, 8 heads
20 trials, 11 heads
50 trials, 27 heads
500 trials, 232 heads
Density
0.0
0.2
0.4
0.6
0.8
1.0
$P(H)$, Probability of Heads


action
reward
Multi-armed Bandit
state
action
reward
Contextual Bandit


A
B
23%
CONTROL
37%
VARIATION


A/B Test
A = 50%
B = 50%
A = 100%
(if A outperformed B)
Time
Week:
1
2
3
4
5
6
A = 33%
B = 33%
C = 33%
A = 40%
B = 30%
C = 30%
A = 50%
B = 25%
C = 25%
A = 30%
B = 30%
C = 40%
A = 25%
B = 25%
C = 50%
A = 20%
D = 20%
C = 60%
Bandit Test