Christof Teuscher

ECE 410/510: Hardware for AI and ML

# Initial steps with project

Portland State University
Department of Electrical and Computer Engineering (ECE)
www.teuscher-lab.com
teuscher@pdx.edu

teuscher Lab teuscher-lab.com

Portland State UNIVERSITY



## Goal: get you started (if you are stuck)

1. Pick an interesting code/workload. E.g., see examples:
   1. QR code recognition
   2. Reinforcement learning (FrozenLake problem)
   3. DeepSpeech (speech-to-text engine, pre-trained)
2. Analyze and benchmark the code to identify possible bottlenecks.
3. Determine total execution time = baseline for HW/SW comparison.

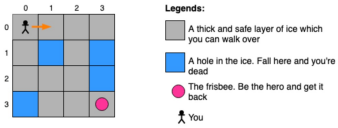
teuscher Lab teuscher-lab.com

Portland State UNIVERSITY



## Scale the problems up to understand better

1. Recognize 1,000, 10,000, 100,000 QR codes
2. Enlarge the FrozenLake grid and place and add a fixed percentage of holes in the ice (e.g., 5%): 5x5, 10x10, 50x50, 100x100, 1,000x1,000 …




teuscher Lab teuscher-lab.com

Portland State UNIVERSITY



## Total execution time (1)

```python
import time

start_time = time.time()

# Code to be timed
def my_function():
    result = 0
    for i in range(1000000):
        result += i
    return result

my_function()

end_time = time.time()
execution_time = end_time - start_time

print(f"Execution time: {execution_time} seconds")
```

Python

```python
import time

start_time = time.perf_counter()

# Code to be timed
def my_function():
    result = 0
    for i in range(1000000):
        result += i
    return result

my_function()

end_time = time.perf_counter()
execution_time = end_time - start_time

print(f"Execution time: {execution_time} seconds")
```

For more precise timing, especially for short durations, `time.perf_counter()` is recommended. It provides the highest available resolution to measure a short duration.

teuscher Lab teuscher-lab.com

Portland State UNIVERSITY



## Total execution time (2)

Python

```python
import timeit

# Code to be timed as a string
code_to_time = """
def my_function():
    result = 0
    for i in range(1000000):
        result += i
    return result
my_function()
"""

execution_time = timeit.timeit(code_to_time, number=1)

print(f"Execution time: {execution_time} seconds")
```

The `timeit` module is specifically designed for measuring the execution time of small code snippets.

teuscher Lab teuscher-lab.com

Portland State UNIVERSITY



## Total execution time (3)

```
[bootis:ece_510_HW_for_AI_and_ML_2025/python/qr] cteuscheN python3 qr2_profile.py qr-code-21x21.png
Execution time 1: 2.1467672119140625e-06 seconds
Execution time 2: 5.489999999888864e-07 seconds
Execution time 3: 6.220000000054736e-07 seconds
[bootis:ece_510_HW_for_AI_and_ML_2025/python/qr] cteuscheN python3 qr2_profile.py qr-code-21x21.png
Execution time 1: 2.1467672119140625e-06 seconds
Execution time 2: 4.089999999770895e-07 seconds
Execution time 3: 5.879999999958141e-07 seconds
[bootis:ece_510_HW_for_AI_and_ML_2025/python/qr] cteuscheN python3 qr2_profile.py qr-code-21x21.png
Execution time 1: 2.861022949218756e-06 seconds
Execution time 2: 2.648999999908566e-06 seconds
Execution time 3: 6.570000001580027e-07 seconds
[bootis:ece_510_HW_for_AI_and_ML_2025/python/qr] cteuscheN python3 qr2_profile.py qr-code-21x21.png
Execution time 1: 9.536743164062e-07 seconds
Execution time 2: 5.140000000070089e-07 seconds
Execution time 3: 5.260000000086926e-07 seconds
[bootis:ece_510_HW_for_AI_and_ML_2025/python/qr] cteuscheN python3 qr2_profile.py qr-code-21x21.png
Execution time 1: 2.1457672119140625e-06 seconds
Execution time 2: 5.240000000174534e-07 seconds
Execution time 3: 5.359999999854279e-07 seconds
[bootis:ece_510_HW_for_AI_and_ML_2025/python/qr] cteuscheN python3 qr2_profile.py qr-code-21x21.png
Execution time 1: 1.907346328125e-06 seconds
Execution time 2: 5.409999998943931e-07 seconds
Execution time 3: 6.000000001725344e-07 seconds
[bootis:ece_510_HW_for_AI_and_ML_2025/python/qr] cteuscheN python3 qr2_profile.py qr-code-21x21.png
Execution time 1: 9.536743164062e-07 seconds
Execution time 2: 4.670000000039926e-07 seconds
Execution time 3: 5.959999999914032e-07 seconds
[bootis:ece_510_HW_for_AI_and_ML_2025/python/qr] cteuscheN python3 qr2_profile.py qr-code-21x21.png
Execution time 1: 1.192092895078125e-06 seconds
Execution time 2: 5.990000000177798e-07 seconds
Execution time 3: 6.559999999877775e-07 seconds
```

teuscher Lab teuscher-lab.com

Portland State UNIVERSITY

# Total execution time (4)

```
# Execution time
start_time1 = time.time()
start_time2 = time.perf_counter()
start_time3 = timeit.default_timer()

repeats = 10

if __name__ == "__main__":
    import sys

for r in range(repeats):

    if len(sys.argv) != 2:
        print("Usage: python qr_recognizer.py <image_path>")
        sys.exit(1)

    image_path = sys.argv[1]
    recognizer = QRCodeRecognizer()

    try:
        result = recognizer.recognize(image_path)
        print("QR Code Recognition Result:")
        print(result)
    except Exception as e:
        print(f"Error: {e}")


end_time1 = time.time()
end_time2 = time.perf_counter()
execution_time1 = end_time1 - start_time1
execution_time2 = end_time2 - start_time2
execution_time3 = timeit.default_timer() - start_time3

print(f"Execution time 1: {execution_time1} seconds")
print(f"Execution time 2: {execution_time2} seconds")
print(f"Execution time 3: {execution_time3} seconds")
```

```
Execution time 1: 1.8990371227264404 seconds
Execution time 2: 1.8990448829999997 seconds
Execution time 3: 1.8990454230000002 seconds
```





# Code analysis (1)

**Static Code Analysis Tools**

**1. Using PyCallGraph**

PyCallGraph is a tool that creates call graph visualizations for Python applications:

```python
# Install with: pip install pycallgraph graphviz
from pycallgraph import PyCallGraph
from pycallgraph.output import GraphvizOutput

def analyze_qr_code():
    with PyCallGraph(output=GraphvizOutput()):
        recognizer = QRCodeRecognizer()
        result = recognizer.recognize("path_to_qr_image.png")
        print(result)

analyze_qr_code()
```

**2. Using pyan for Static Analysis**

Pyan analyzes Python code and generates approximate call graphs:

```python
# Install with: pip install pyan
# Usage from command line
# pyan qr_recognizer.py --dot > qr_graph.dot
# dot -Tpng qr_graph.dot -o qr_graph.png
```





# Code analysis (2)

**Runtime Analysis Tools**

**3. Using cProfile with Visualization**

```python
import cProfile
import pstats
from pstats import SortKey

# Profile the execution
cProfile.run('recognizer.recognize("path_to_qr_image.png")', '

# Analyze the results
p = pstats.Stats('qr_stats')
p.strip_dirs().sort_stats(SortKey.CUMULATIVE).print_stats(10)

# For visualization, you can use gprof2dot
# pip install gprof2dot
# gprof2dot -f pstats qr_stats | dot -Tpng -o qr_profile.png
```

**4. Using SnakeViz for Interactive Visualization**

```python
# Install with: pip install snakeviz
# Run profiling
import cProfile
cProfile.run('recognizer.recognize("path_to_qr_image.png")', '

# Then from command line:
# snakeviz qr_stats
```





# Code analysis (3)

**Performance Visualization**

**6. Visualize Performance Bottlenecks**

To visualize execution time of different components:

```python
import time
import numpy as np
import matplotlib.pyplot as plt

def benchmark_qr_components(image_path, iterations=10):
    recognizer = QRCodeRecognizer()
    components = {}

    # Benchmark image loading
    start = time.time()
    for _ in range(iterations):
        img = recognizer.load_image(image_path)
    components["load_image"] = (time.time() - start) / iterati

    # Benchmark thresholding
    start = time.time()
```

**Using Specialized Tools**

For more advanced analysis, consider these specialized tools:

1. **Doxygen** - Can generate call graphs if properly configured
2. **CodeViz** - Works with C/C++ extensions but can be used on some Python projects
3. **Memray** - For memory profiling with visualization capabilities
4. **Py-Spy** - For sampling profiler that can generate flame graphs

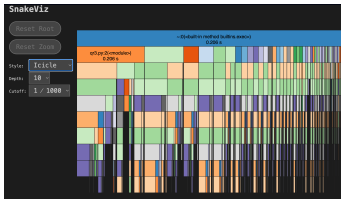
Each of these methods offers different insights into your code's structure and performance. Combining a few of them will give you the most comprehensive understanding of your QR code recognizer's data and control flow.

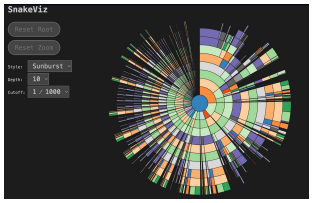




# cProfile and SnakeViz

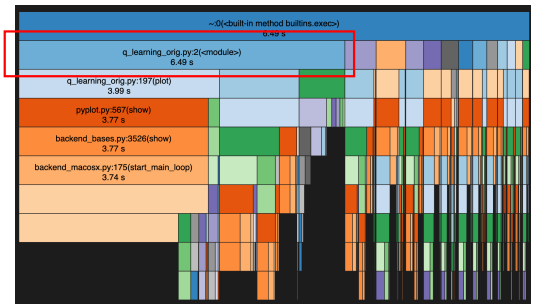
1. `pip install snakeviz`
2. `python3 -m cProfile -o output.prof qr_recognizer.py qr.png`
3. `snakeviz output.prof`







| ncalls | tottime | percall | cumtime | percall | filename:lineno(function) |
|---|---|---|---|---|---|
| 37/36 | 0.03897 | 0.001082 | 0.04111 | 0.001142 | ~:0(<built-in method _imp.create_dynamic>) |
| 153 | 0.01551 | 0.0001014 | 0.01551 | 0.0001014 | ~:0(<built-in method marshal.loads>) |
| 1849 | 0.01443 | 7.803e-06 | 0.01443 | 7.803e-06 | qt3.py:105(_is_potential_finder_pattern) |
| 165 | 0.008252 | 5.001e-05 | 0.008252 | 5.001e-05 | ~:0(<method 'read' of '_io.BufferedReader' objects>) |
| 153 | 0.006271 | 4.099e-05 | 0.006271 | 4.099e-05 | ~:0(<built-in method io.open_code>) |
| 809 | 0.005534 | 6.84e-06 | 0.005534 | 6.84e-06 | ~:0(<built-in method posix.stat>) |
| 390/389 | 0.00524 | 1.347e-05 | 0.02292 | 5.893e-05 | ~:0(<built-in method builtins.__build_class__>) |
| 20 | 0.004723 | 0.0002362 | 0.008198 | 0.0004099 | enum.py:179(__new__) |
| 1 | 0.003492 | 0.003492 | 0.005462 | 0.005462 | histograms.py:678(histogram) |
| 388 | 0.002841 | 7.323e-06 | 0.01371 | 3.534e-05 | <frozen importlib._bootstrap_external>:1438(find_spec) |
| 65 | 0.00221 | 3.4e-05 | 0.00221 | 3.4e-05 | ~:0(<built-in method posix.getcwd>) |

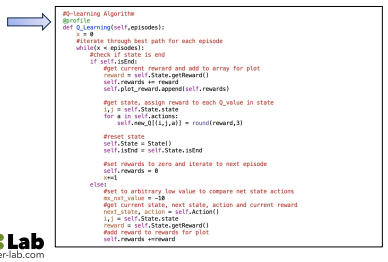
- **ncalls:** Total number of calls to the function. If there are two numbers, that means the function recursed and the first is the total number of calls and the second is the number of primitive (non-recursive) calls.
- **tottime:** Total time spent in the function, not including time spent in calls to sub-functions
- **percall:** `tottime` divided by `ncalls`
- **cumtime:** Cumulative time spent in this function and all sub-functions
- **percall:** `cumtime` divided by `ncalls`
- **filename:lineno(function):** File name and line number were the function is defined, and the function's name





# line_profiler

1. `kernprof -l q_learning_profiling.py`
2. `python3 -m line_profiler -rmt "q_learning_profiling.py.lprof"`



```
#Q-learning Algorithm
@profile
def Q_Learning(self,episodes):
    x = 0
    #iterate through best path for each episode
    while(x < episodes):
        #check if state is end
        if self.isEnd:
            #get current reward and add to array for plot
            reward = self.State.getReward()
            self.rewards += reward
            self.plot_reward.append(self.rewards)

            #get state, assign reward to each Q_value in state
            i,j = self.State.state
            for a in self.actions:
                self.new_Q[(i,j,a)] = round(reward,3)

            #reset state
            self.State = State()
            self.isEnd = self.State.isEnd

            #set rewards to zero and iterate to next episode
            self.rewards = 0
            x+=1
        else:
            #set to arbitrary low value to compare net state actions
            mx_nxt_value = -10
            #get current state, next state, action and current reward
            next_state, action = self.Action()
            i,j = self.State.state
            reward = self.State.getReward()
            #add reward to rewards for plot
            self.rewards +=reward
```





```
Total time: 1.91771 s
File: q_learning_profiling.py
Function: Q_Learning at line 141

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
   141                                           @profile
   142                                           def Q_Learning(self,episodes):
   143         1          1.0      1.0      0.0      x = 0
   144                                               #iterate through best path for each episode
   145     94259      24463.0      0.3      1.3      while(x < episodes):
   146                                                   #check if state is end
   147     94258      22655.0      0.2      1.2          if self.isEnd:
   148                                                       #get current reward and add to array for plot
   149     10000      13540.0      1.4      0.7              reward = self.State.getReward()
   150     10000       3810.0      0.4      0.2              self.rewards += reward
   151     10000       5266.0      0.5      0.3              self.plot_reward.append(self.rewards)
   152                                                   
   153                                                   #get state, assign reward to each Q_value in state
   154     10000       3039.0      0.3      0.2          i,j = self.State.state
   155     50000      12719.0      0.3      0.7          for a in self.actions:
   156     40000      26152.0      0.7      1.4              self.new_Q[(i,j,a)] = round(reward,3)
   157                                                   
   158                                                   #reset state
   159     10000       9741.0      1.0      0.5          self.State = State()
   160     10000       3885.0      0.4      0.2          self.isEnd = self.State.isEnd
   161                                                   
   162                                                   #set rewards to zero and iterate to next episode
   163     10000       2903.0      0.3      0.2          self.rewards = 0
   164     10000       3693.0      0.4      0.2          x+=1
```





```
Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
   165                                                       else:
   166                                                           #set to arbitrary low value to compare net state actions
   167     84258      19736.0      0.2      1.0                  mx_nxt_value = -10
   168                                                           #get current state, next state, action and current reward
   169     84258     528816.0      6.3     27.6                  next_state, action = self.Action()
   170     84258      25683.0      0.3      1.3                  i,j = self.State.state
   171     84258     108610.0      1.3      5.7                  reward = self.State.getReward()
   172                                                           #add reward to rewards for plot
   173     84258      33549.0      0.4      1.7                  self.rewards +=reward
   174                                                   
   175                                                           #iterate through actions to find max Q value for action based on next state action
   176    421290     115314.0      0.3      6.0                  for a in self.actions:
   177    337032     111121.0      0.3      5.8                      nxtStateAction = (next_state[0], next_state[1], a)
   178    337032     272470.0      0.8     14.2                      q_value = (1-self.alpha)*self.Q[(i,j,action)] + self.alpha*(reward + self.gamma*…
   179                                                   
   180                                                               #find largest Q value
   181    337032      87482.0      0.3      4.6                      if q_value >= mx_nxt_value:
   182    212265      47640.0      0.2      2.5                          mx_nxt_value = q_value
   183                                                   
   184                                                           #next state is now current state, check if end state
   185     84258      95663.0      1.1      5.0                  self.State = State(state=next_state)
   186     84258     112622.0      1.3      5.9                  self.State.isEndFunc()
   187     84258      29997.0      0.4      1.6                  self.isEnd = self.State.isEnd
   188                                                   
   189                                                           #update Q values with max Q value for next state
   190     84258     100733.0      1.2      5.3                  self.new_Q[(i,j,action)] = round(mx_nxt_value,3)
   191                                                   
   192                                                       #copy new Q values to Q table
   193     94258      96267.0      1.0      5.0              self.Q = self.new_Q.copy()
   194                                               #print final Q table output
   195         1        144.0    144.0      0.0      print(self.Q)
```





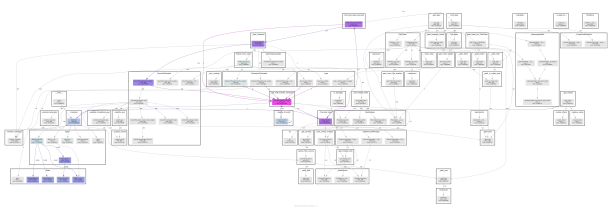
```
Total time: 0.828347 s
File: q_learning_profiling.py
Function: Action at line 111

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
   111                                           @profile
   112                                           def Action(self):
   113                                               #random value vs epsilon
   114     83741      27544.0      0.3      3.3      rnd = random.random()
   115                                               #set arbitrary low value to compare with Q values to find max
   116     83741      19866.0      0.2      2.3      mx_nxt_reward =-10
   117     83741      17424.0      0.2      2.1      action = None
   118                                               
   119                                               #9/10 find max Q value over actions
   120     83741      24591.0      0.3      3.0      if(rnd >self.epsilon) :
   121                                                   #iterate through actions, find Q  value and choose best
   122    376785      94958.0      0.3     11.5          for k in self.actions:
   123                                                   
   124    301428      80642.0      0.3      9.7              i,j = self.State.state
   125                                                   
   126    301428     108115.0      0.4     13.1              nxt_reward = self.Q[(i,j, k)]
   127                                                   
   128    301428      73107.0      0.2      8.8              if nxt_reward >= mx_nxt_reward:
   129    180726      37599.0      0.2      4.5                  action = k
   130    180726      35654.0      0.2      4.3                  mx_nxt_reward = nxt_reward
   131                                                   
   132                                               #else choose random action
   133                                               else:
   134      8384     170774.0     20.4     20.6          action = np.random.choice(self.actions)
   135                                               
   136                                               #select the next state based on action chosen
   137     83741     118470.0      1.4     14.3      position = self.State.nxtPosition(action)
   138     83741      20403.0      0.2      2.5      return position,action
```

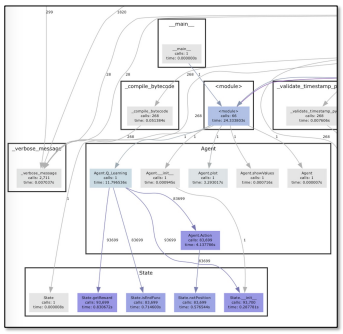




# Pycallgraph2: Dynamic Call Graph Generation

1. Requires `pycallgraph2` and `graphviz`
2. `pycallgraph graphviz --output-file output.png q_learning_orig.py`





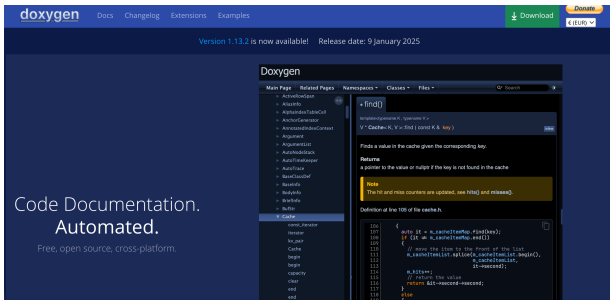




| ncalls | tottime | percall | cumtime | percall | filename:lineno(function) |
|---|---|---|---|---|---|
| 1 | 3.438 | 3.438 | 3.743 | 3.743 | ~:0(<built-in method matplotlib.backends._macosx.show>) |
| 46/44 | 0.7926 | 0.01801 | 0.7994 | 0.01817 | ~:0(<built-in method _imp.create_dynamic>) |
| 1 | 0.4153 | 0.4153 | 0.9049 | 0.9049 | q_learning_orig.py:141(Q_Learning) |
| 83522 | 0.1256 | 1.504e-06 | 0.2928 | 3.506e-06 | q_learning_orig.py:111(Action) |
| 292 | 0.09859 | 0.0003376 | 0.09859 | 0.0003376 | ~:0(<method 'read' of '_io.BufferedReader' objects>) |
| 1 | 0.07742 | 0.07742 | 0.1346 | 0.1346 | backend_bases.py:2673(create_with_canvas) |
| 7178 | 0.06007 | 8.369e-06 | 0.107 | 1.491e-05 | inspect.py:626(cleandoc) |
| 8403 | 0.05586 | 6.647e-06 | 0.1172 | 1.394e-05 | ~:0(<method 'choice' of 'numpy.random.mtrand.RandomState' objects>) |
| 123564 | 0.055 | 4.451e-07 | 0.055 | 4.451e-07 | ~:0(<built-in method builtins.round>) |
| 292 | 0.04718 | 0.0001616 | 0.04718 | 0.0001616 | ~:0(<built-in method marshal.loads>) |
| 115 | 0.04713 | 0.0004098 | 0.04713 | 0.0004098 | ~:0(<method 'set_message' of 'matplotlib.backends._macosx.NavigationToolba |
| 93522 | 0.04602 | 4.921e-07 | 0.04602 | 4.921e-07 | q_learning_orig.py:33(getReward) |
| 7 | 0.04531 | 0.006473 | 0.04541 | 0.006488 | ~:0(<method 'draw_path' of 'matplotlib.backends._backend_agg.RendererAgg' |
| 83522 | 0.04079 | 4.884e-07 | 0.04079 | 4.884e-07 | q_learning_orig.py:53(nxtPosition) |





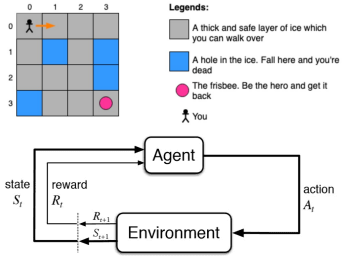



https://www.doxygen.nl





# Q learning (1)

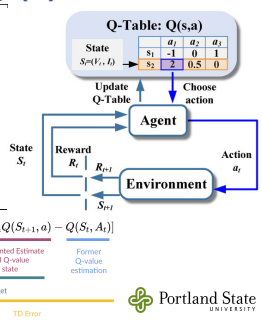







# Q learning (2)

Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
  Initialize $s$
  Repeat (for each step of episode):
    Choose $a$ from $s$ using policy derived from $Q$
    Take action $a$, observe $r$, $s'$
    Update
    $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
    $s \leftarrow s'$;
  Until s is terminal




$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

New Q-value estimation | Former Q-value estimation | Learning Rate | Immediate Reward | Discounted Estimate optimal Q-value of next state | Former Q-value estimation

TD Target

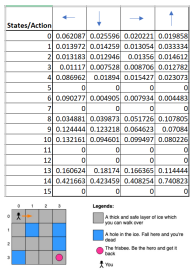
TD Error

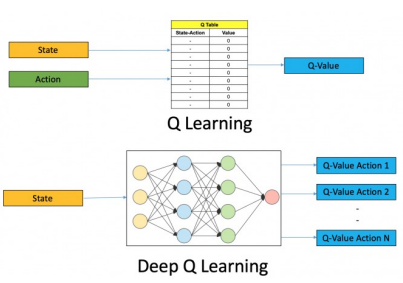




# Q learning (3)

| States/Action | ← | ↓ | → | ↑ |
|---|---|---|---|---|
| 0 | 0.062087 | 0.025596 | 0.020221 | 0.019858 |
| 1 | 0.013972 | 0.014259 | 0.013054 | 0.033334 |
| 2 | 0.013183 | 0.012946 | 0.01356 | 0.014612 |
| 3 | 0.01117 | 0.007528 | 0.008706 | 0.012782 |
| 4 | 0.086962 | 0.01894 | 0.015427 | 0.023073 |
| 5 | 0 | 0 | 0 | 0 |
| 6 | 0.090277 | 0.004905 | 0.007934 | 0.004483 |
| 7 | 0 | 0 | 0 | 0 |
| 8 | 0.034881 | 0.039873 | 0.051726 | 0.107805 |
| 9 | 0.124444 | 0.123218 | 0.064023 | 0.07084 |
| 10 | 0.132161 | 0.094601 | 0.099497 | 0.080226 |
| 11 | 0 | 0 | 0 | 0 |
| 12 | 0 | 0 | 0 | 0 |
| 13 | 0.160624 | 0.18174 | 0.166365 | 0.114444 |
| 14 | 0.421663 | 0.423459 | 0.408254 | 0.740823 |
| 15 | 0 | 0 | 0 | 0 |





Christof Teuscher • teuscher@pdx.edu



# Efficient HW Q learning implementation

IEEE Access

An Efficient Hardware Implementation of Reinforcement Learning: The Q-Learning Algorithm


FIGURE 1. Reinforcement learning framework.

The agent receives the state $s_{t+1}$ and the reward $r_{t+1}$ from the observer, while the next action is generated by the PG according to the values of the Q-Matrix stored into the Q-Learning accelerator.

Note that $s_t$, $a_t$ and $r_t$ are obtained by delaying $s_{t+1}$, $a_{t+1}$ and $r_{t+1}$ by means of registers. $s_t$ and $a_t$ represent the indices of the rows and columns of the Q-Matrix, respectively. These delays do not affect the convergence of the Q-Learning algorithm, as proved in [30].

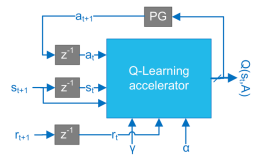



FIGURE 2. High level harchitecture of the Q-Learning agent.

$z^{-1}$ = delay registers

https://doi.org/10.1109/ACCESS.2019.2961174



Christof Teuscher • teuscher@pdx.edu

www.teuscher-lab.com/teaching

Efficient HW Q learning implementation

Figure 3 shows the Q-Learning accelerator.

The Q-Matrix is stored into $Z$ Dual-Port RAMs, named Action RAMs. Consequently, we have one memory block per action. Each RAM contains an entire column of the Q-Matrix and the number of memory locations corresponds to the number of states $N$. The read address is the next state $s_{t+1}$, while the write address is the current state $s_t$. The enable signals for the Action RAMs, generated by a decoder driven by the current action $a_t$, select the value $Q(s_t, a_t)$ to be updated. The Action RAMs outputs correspond to a row of the Q-Matrix $Q(s_{t+1}, A)$.

The signal $Q(s_t, a_t)$ is obtained by delaying the output of the memory blocks and then selecting the Action RAM through a multiplexer driven by $a_t$. A MAX block fed by the output of the Action RAMs generates max $Q(s_{t+1}, A)$.

The Q-Updater (Q-Upd) block implements the Q-Matrix update equation (1) generating $Q_{new}(s_t, a_t)$ to be stored into the corresponding Action RAM.

The accelerator can be also used for Deep Q-Learning [23] applications if the Action RAMs are replaced with Neural Network-based approximators.

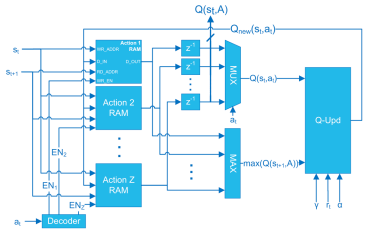



FIGURE 3. Q-Learning accelerator architecture.

https://doi.org/10.1109/ACCESS.2019.2961174

teuscher Lab teuscher-lab.com

Portland State UNIVERSITY

Christof Teuscher • teuscher@pdx.edu

www.teuscher-lab.com/teaching

Efficient HW Q learning implementation

$$Q_{new}(s_t, a_t) = (1-\alpha)Q(s_t, a_t) + \alpha\left(r_t + \gamma \max_a Q(s_{t+1}, a)\right) \quad (1)$$

The variables in (1) refer to:

- $s_t$ and $s_{t+1}$: current and next state of the environment.
- $a_t$ and $a_{t+1}$: current and next action chosen by the agent (according to its policy).
- $\gamma$: discount factor, $\gamma \in [0, 1]$. It defines how much the agent has to take into account long-run rewards instead of immediate ones.
- $\alpha$: learning rate, $\alpha \in [0, 1]$. It determines how much the newest piece of knowledge has to replace the older one.
- $r_t$: current reward value.

How many multipliers?

https://doi.org/10.1109/ACCESS.2019.2961174

teuscher Lab teuscher-lab.com

Portland State UNIVERSITY

Christof Teuscher • teuscher@pdx.edu

www.teuscher-lab.com/teaching

Efficient HW Q learning implementation

$$Q_{new}(s_t, a_t) = (1-\alpha)Q(s_t, a_t) + \alpha\left(r_t + \gamma \max_a Q(s_{t+1}, a)\right) \quad (1)$$

The variables in (1) refer to:

- $s_t$ and $s_{t+1}$: current and next state of the environment.
- $a_t$ and $a_{t+1}$: current and next action chosen by the agent (according to its policy).
- $\gamma$: discount factor, $\gamma \in [0, 1]$. It defines how much the agent has to take into account long-run rewards instead of immediate ones.
- $\alpha$: learning rate, $\alpha \in [0, 1]$. It determines how much the newest piece of knowledge has to replace the older one.
- $r_t$: current reward value.

### B. Q-UPDATER BLOCK

Equation (1) can be rearranged as

$$Q_{new}(s_t, a_t) = Q(s_t, a_t) + \alpha\left(r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right) \quad (2)$$

How many multipliers?

to obtain an efficient implementation. Equation (2) is computed by using 2 multipliers, while (1) requires 3 multipliers.

The Q-Updater block in Fig. 5 is used to compute (2), generating $Q_{new}(s_t, a_t)$.

The critical path consists in 2 multipliers and 2 adders. In the next section (II-B1) a method to reduce the hardware complexity for the multipliers is illustrated.

https://doi.org/10.1109/ACCESS.2019.2961174

teuscher Lab teuscher-lab.com

Portland State UNIVERSITY

Christof Teuscher • teuscher@pdx.edu

www.teuscher-lab.com/teaching

Efficient HW Q learning implementation

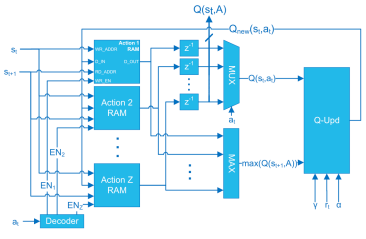



FIGURE 3. Q-Learning accelerator architecture.

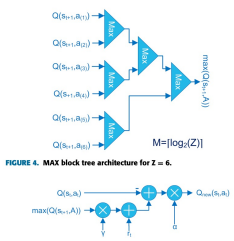



FIGURE 4. MAX block tree architecture for Z = 6.




FIGURE 5. Q-matrix updater block architecture.

https://doi.org/10.1109/ACCESS.2019.2961174

teuscher Lab teuscher-lab.com

Portland State UNIVERSITY

Christof Teuscher • teuscher@pdx.edu

www.teuscher-lab.com/teaching

Efficient HW Q learning implementation

#### 1) APPROXIMATED MULTIPLIERS

The main speed limitation in the updater block is the propagation delay of the multipliers. Using a similar approach to [32], it is possible to replace the full multipliers shown in Fig. 5 with approximated multipliers based on barrel shifters [33]. In this way, we are approximating $\alpha$ and $\gamma$ with a number equal to their nearest power of two (single shifter), or to the nearest sum of powers of two (two or more shifters). Due to the fact that $\alpha$, $\gamma \in [0, 1]$, only right shifts have been used.

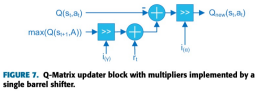



FIGURE 7. Q-Matrix updater block with multipliers implemented by a single barrel shifter.

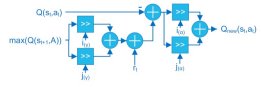



FIGURE 8. Q-Matrix updater block with multipliers implemented by two barrel shifters.

https://doi.org/10.1109/ACCESS.2019.2961174

teuscher Lab teuscher-lab.com

Portland State UNIVERSITY