Approximate Q-Learning

Eirik Fagerhaug
Norwegian University of Science and Technology

Slides Adapted from/based on:

- Slides by Chris G. Willocks
- Grokking DRL (Miguel Morales)
- AI: A Modern Approach (Russel and Norvig)

Reading Material

Russel and Norvig;

- Chapter 23.4.(1,2,3)
- Chapter 22.1.1

Wiki (Assignments)

- Recap
- Generalization in Reinforcement Learning
- Artificial Neural Networks
- Bellman Update for ANNs
- Deep Q-Network

Q-Function

(Action-utility function)

The Q-function as a Bellman Equation:

$$Q(s,a) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma \max_{a'} Q(s',a')]$$

Optimal policy w/r the Q-function:

$$\pi^*(s) = \arg\max_a Q(s,a)$$

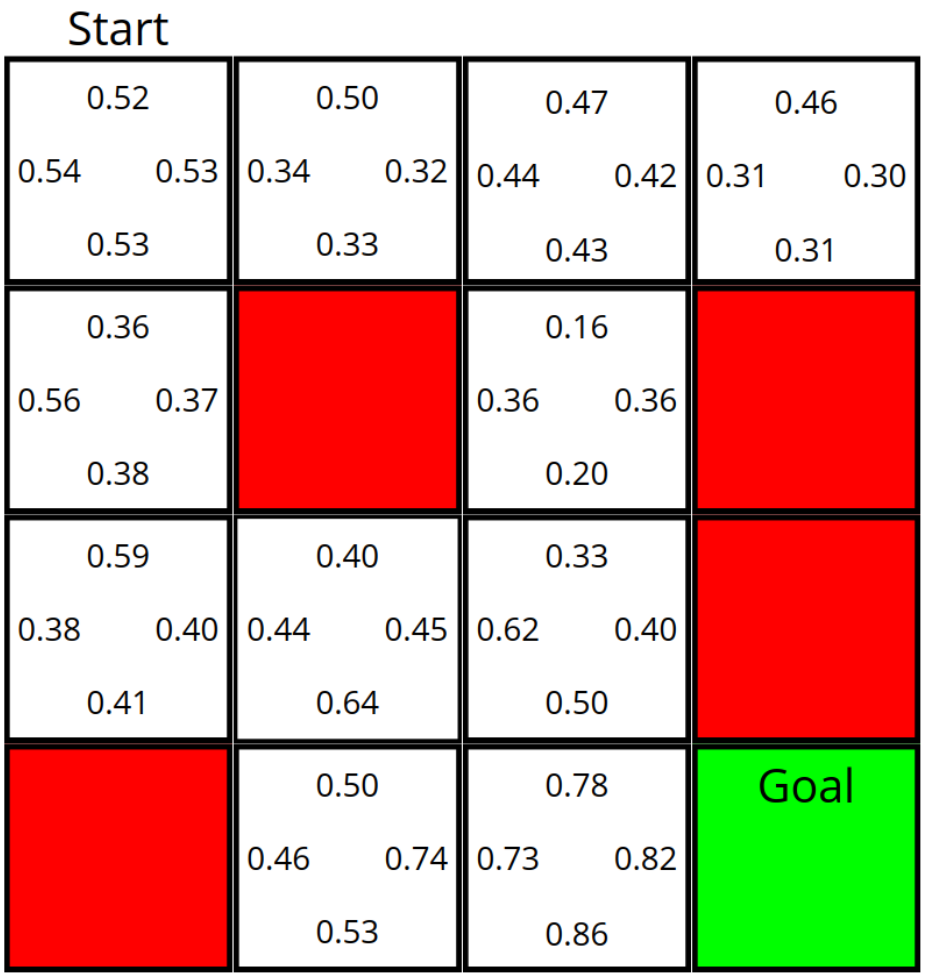
Frozen Lake Environment

Start 0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	Goal +1 15

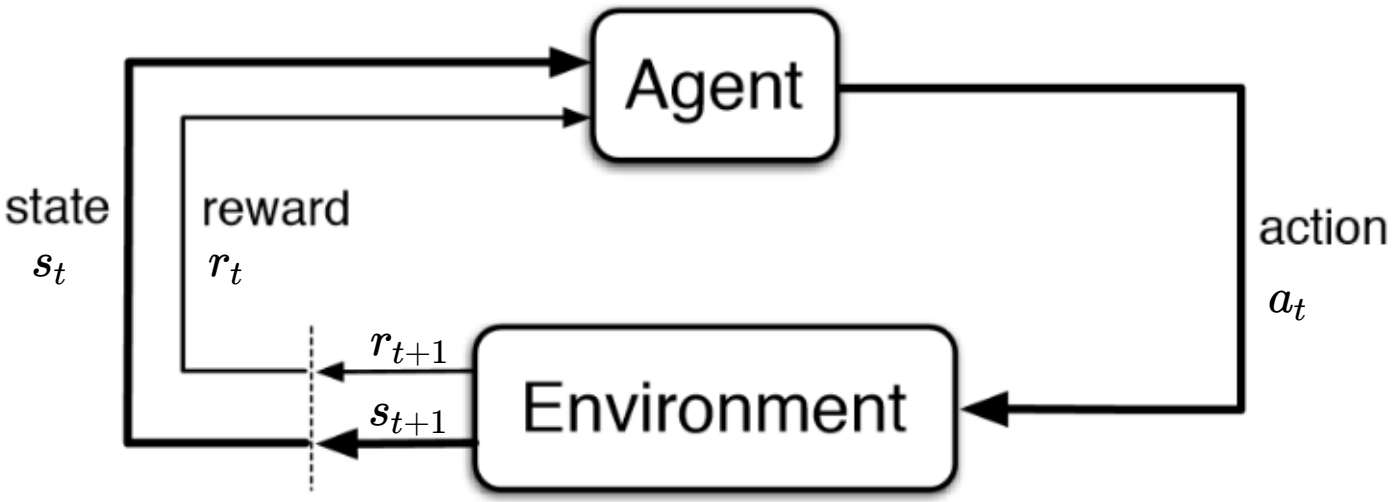
Expected Utility

$$\max_a Q(s, a)$$

Q-Values


Start
0.52
0.54
0.53
0.53
0.50
0.34
0.32
0.33
0.47
0.44
0.42
0.43
0.46
0.31
0.30
0.31
0.36
0.56
0.37
0.38
0.16
0.36
0.36
0.20
0.59
0.38
0.40
0.41
0.40
0.44
0.45
0.64
0.33
0.62
0.40
0.50
0.50
0.46
0.74
0.53
0.78
0.73
0.82
0.86
Goal

Agent-Environment interaction



- At each step *t* the agent:
  - Receives state s_t (and reward r_t)
  - Executes action a_t
- The environment:
  - Receives action a_t
  - Emits state s_{t+1} (and reward r_{t+1})

Bellman Update

$$Q(s,a) \leftarrow Q(s,a) + \alpha[R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$

Q-tables for large state- and action-spaces

For a game with 200x200 pixels, where we use the pixels as the state, we have:

$$|S| = ((255)^3)^{200x200} \approx 6.6 \cdot 10^{288784}$$

(There are between 10^{78} and 10^{82} atoms in the observable universe)

- Recap

→ • Generalization in Reinforcement Learning

- Artificial Neural Networks
- Bellman Update for ANNs
- Deep Q-Network

Generalization in Reinforcement Learning

Problem with large state spaces:

- Too large to save in a Q-table
- Cannot possibly visit all possible states

Solution:

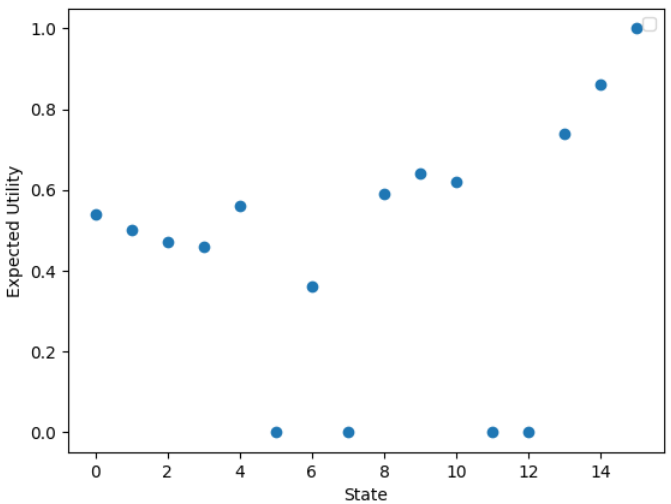
Function approximation; a process of constructing a compact approximation of the true utility- or Q- function.

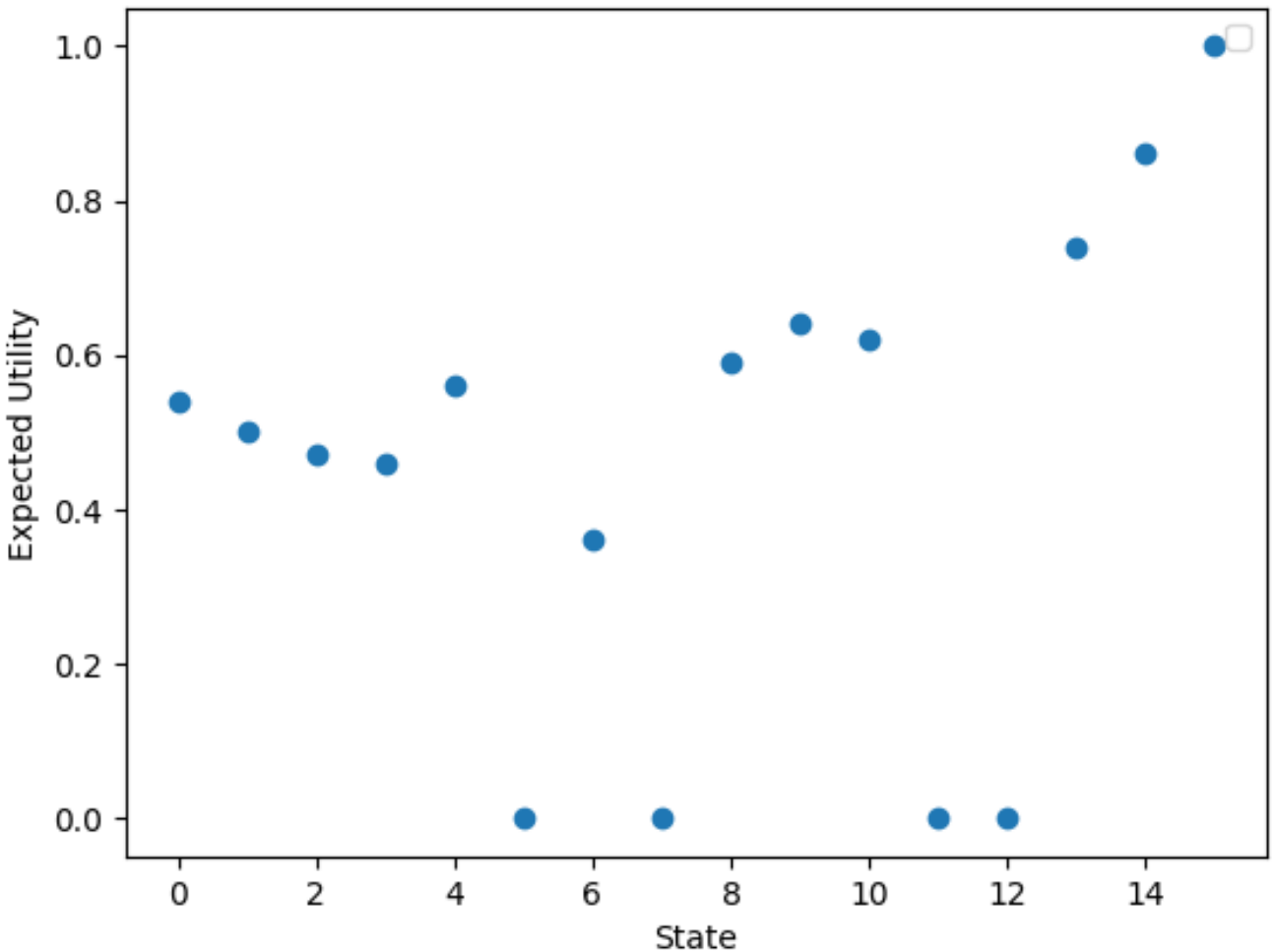
Example; A weighted linear combination of features

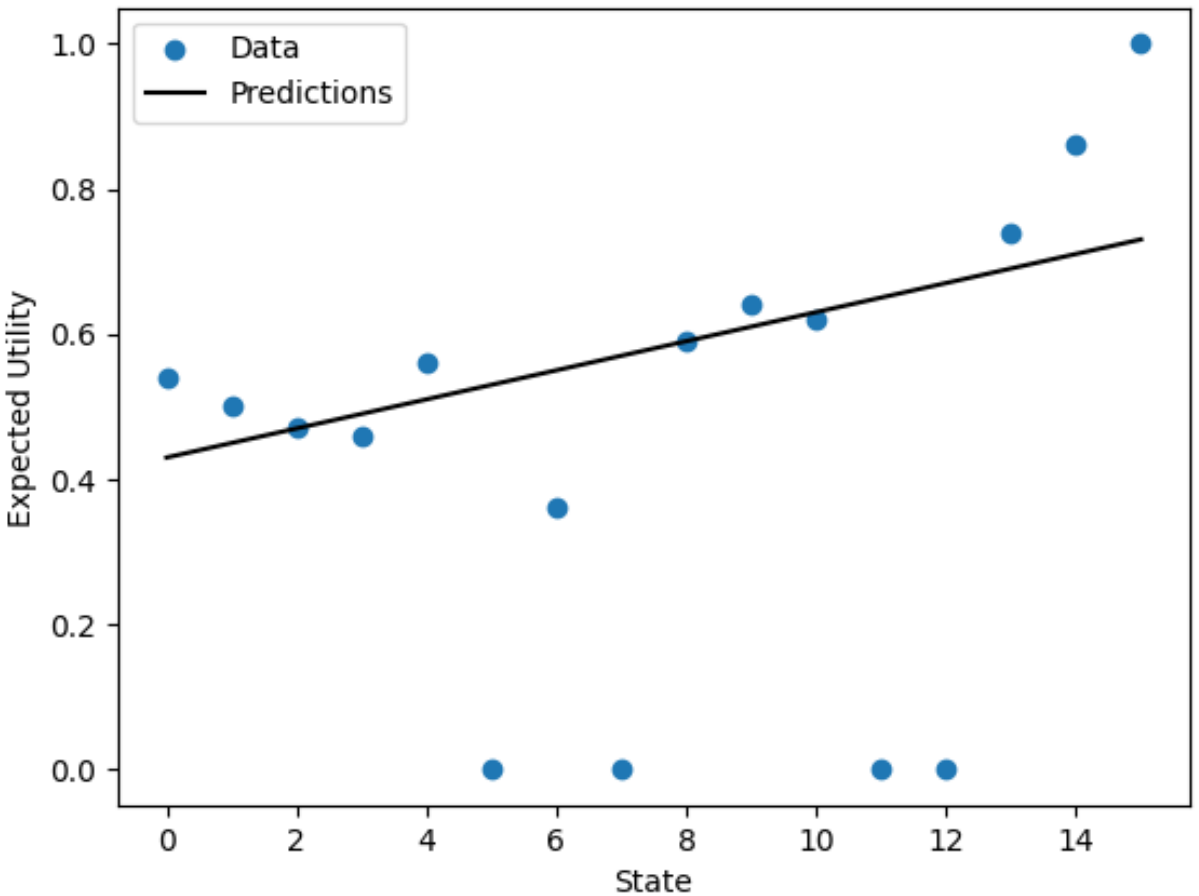
$$\hat{U}_w(s) = w_1 f_1(s) + w_2 f_2(s) + ... + w_n f_n(s)$$

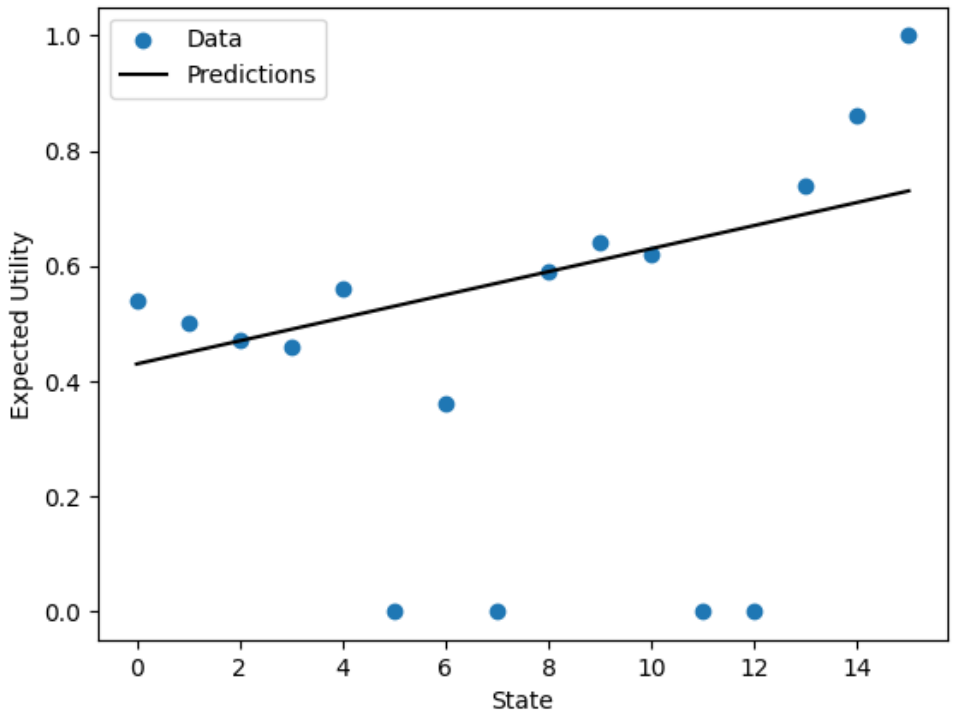
$$\max_a Q(s, a)$$

Start 0.54 0	0.50 1	0.47 2	0.46 3
0.56 4	5	0.36 6	7
0.59 8	0.64 9	0.62 10	11
12	0.74 13	0.86 14	Goal 15 +1




Expected Utility
State
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10
12
14


Data
Predictions
Expected Utility
State
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10
12
14



$$\hat{U}_w(x) = w_0 + w_1 x$$

$$\hat{U}_w(x) = 0.43 + 0.02x$$

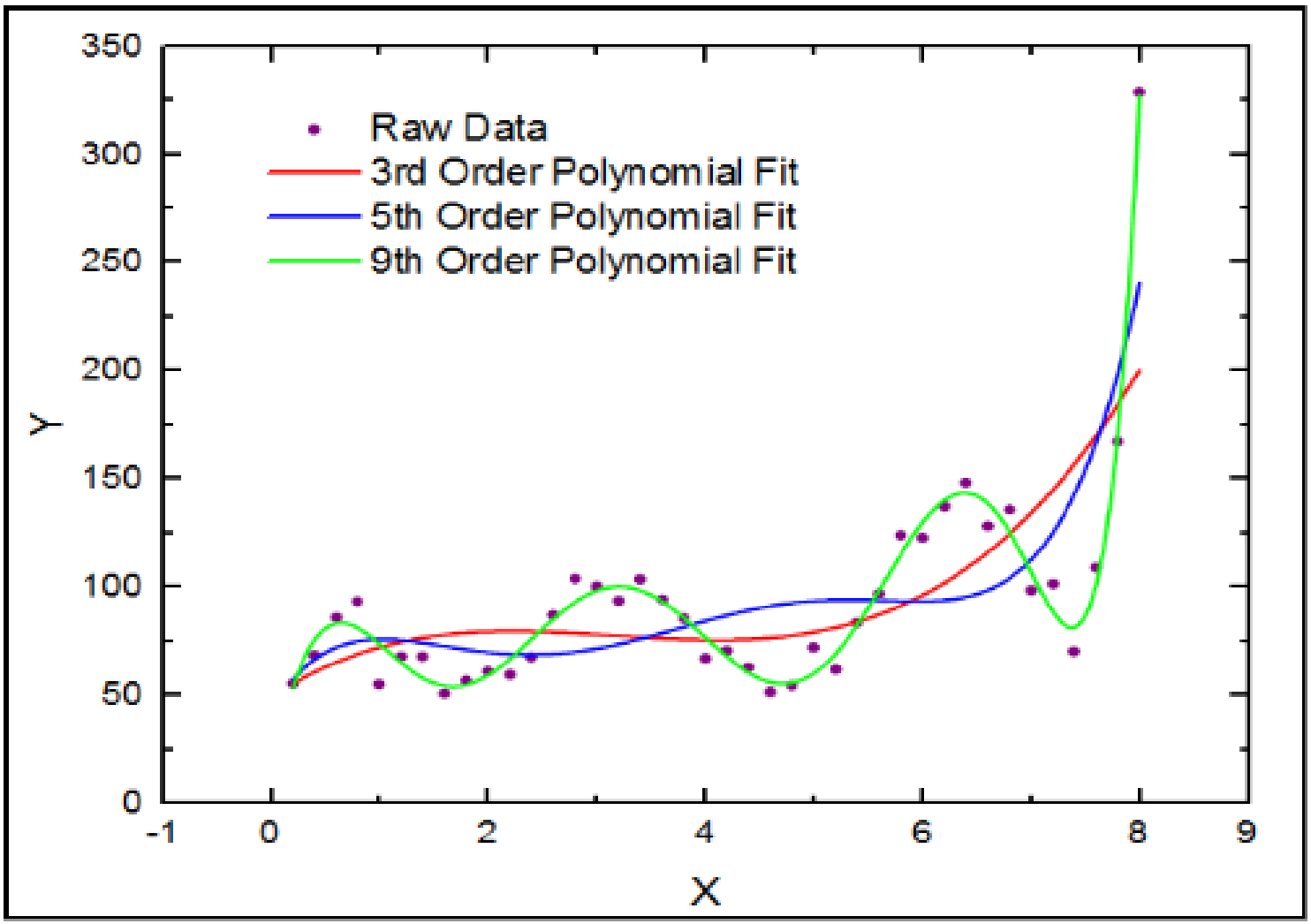


image from WikiMedia by Shobha

- Recap
- Generalization in Reinforcement Learning
- Artificial Neural Networks
- Bellman Update for ANNs
- Deep Q-Network

Artificial Neural Network

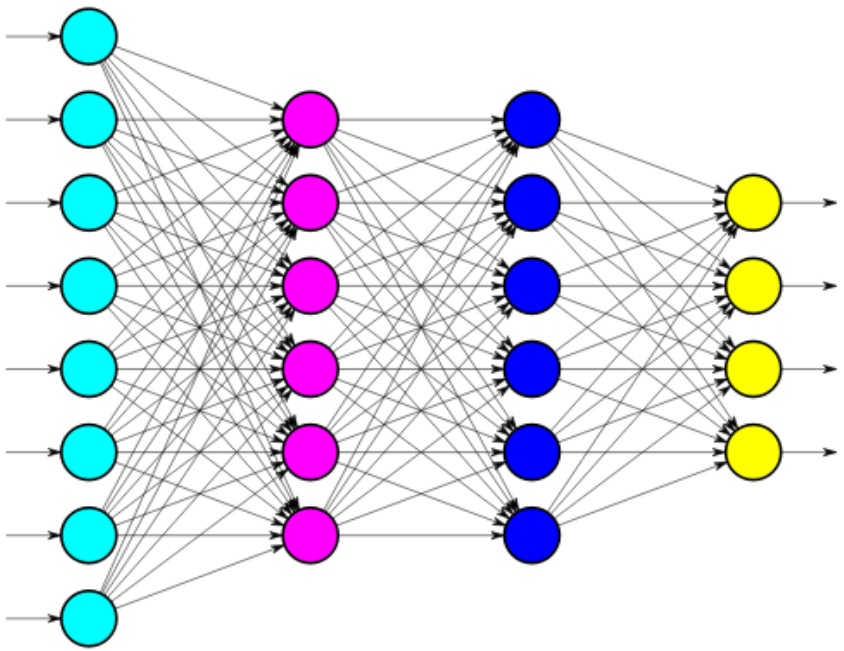
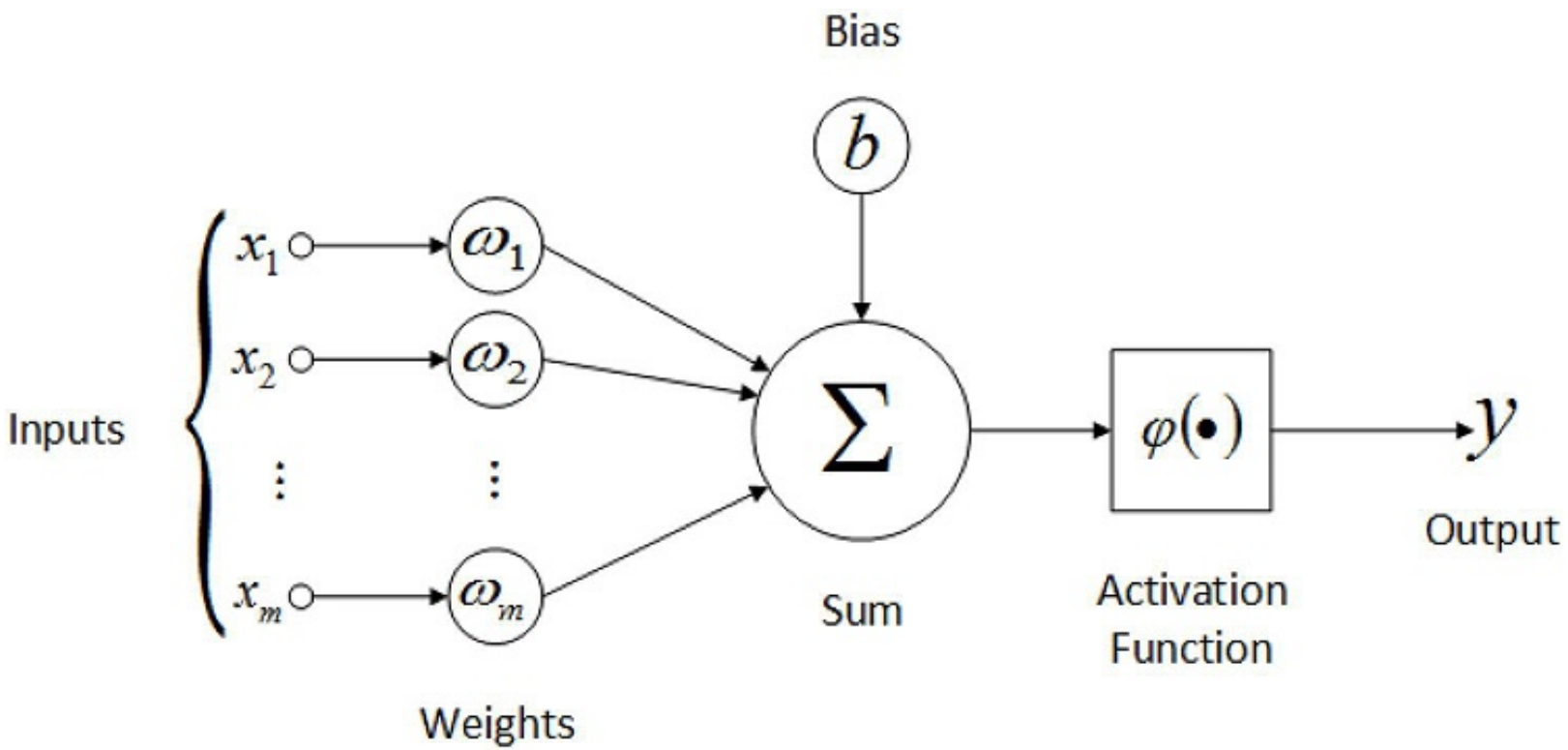
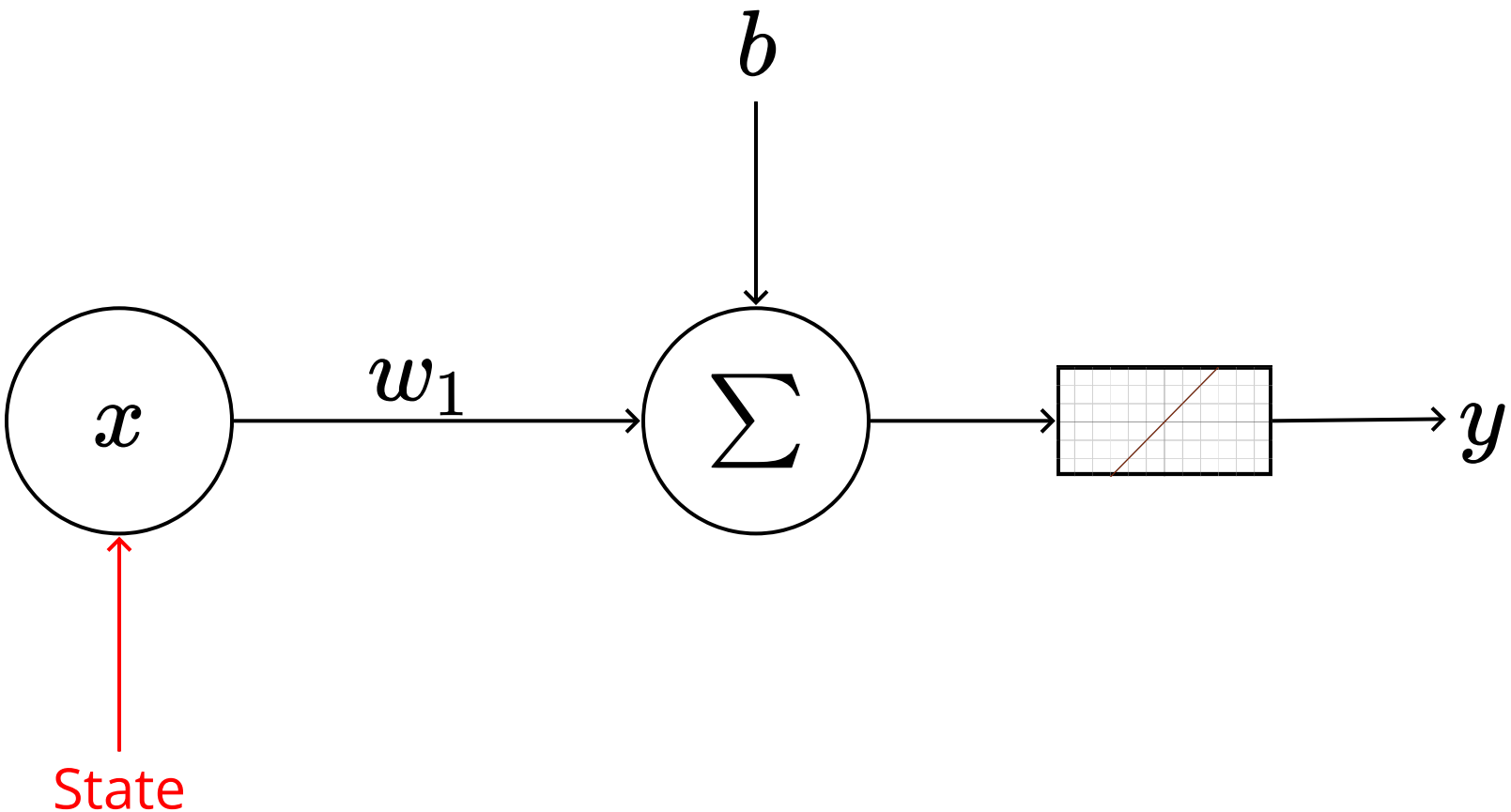
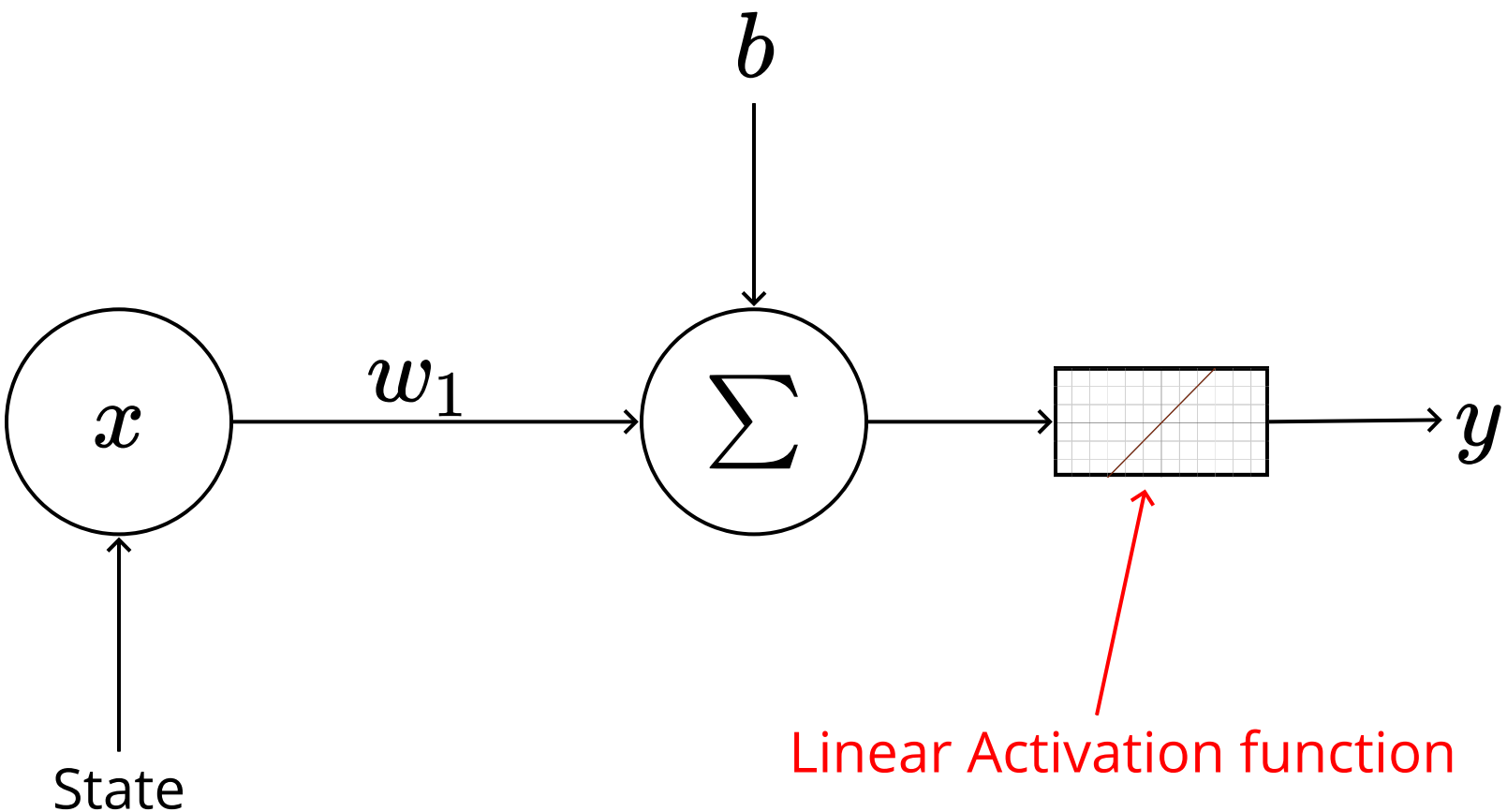
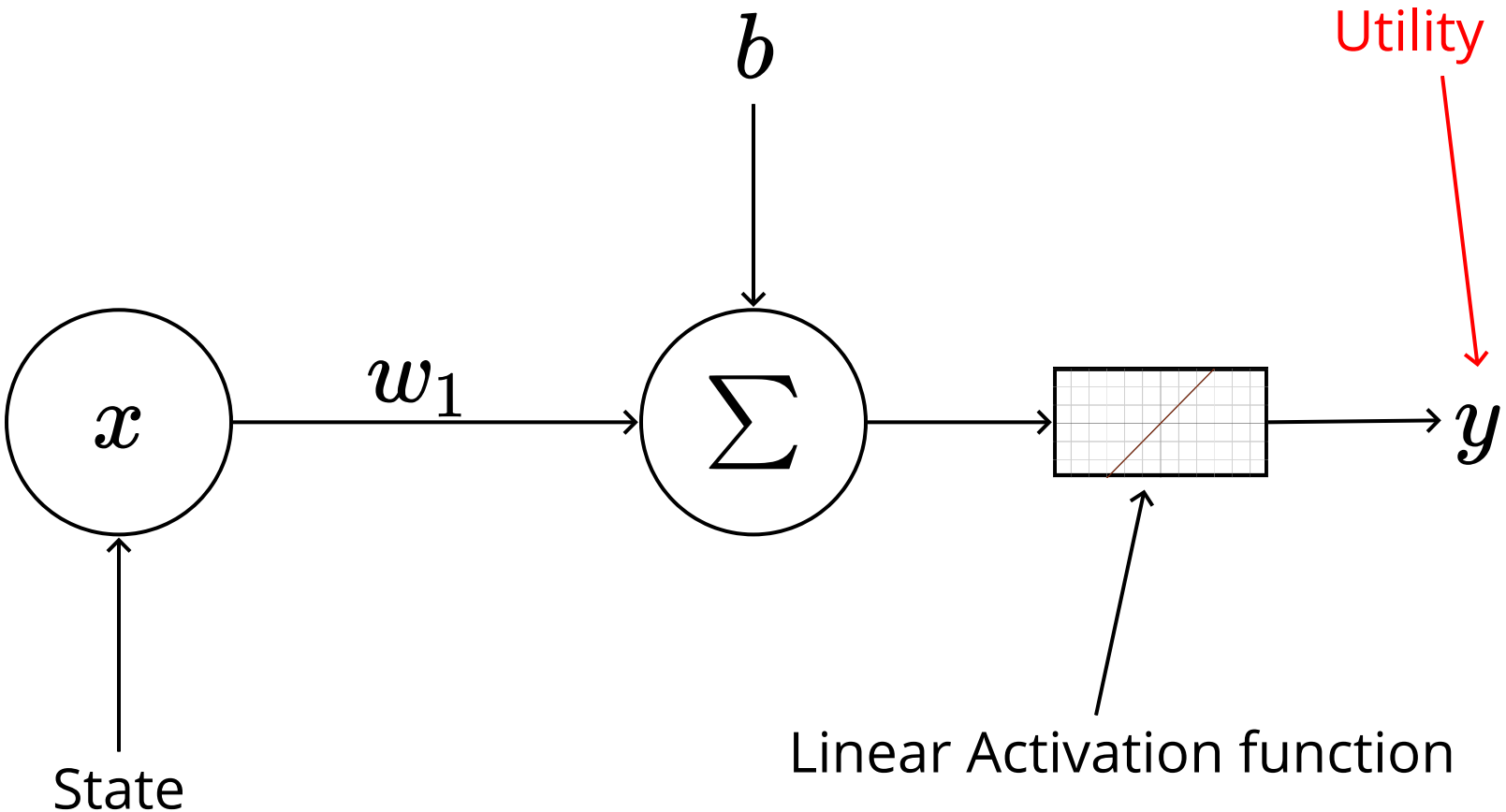


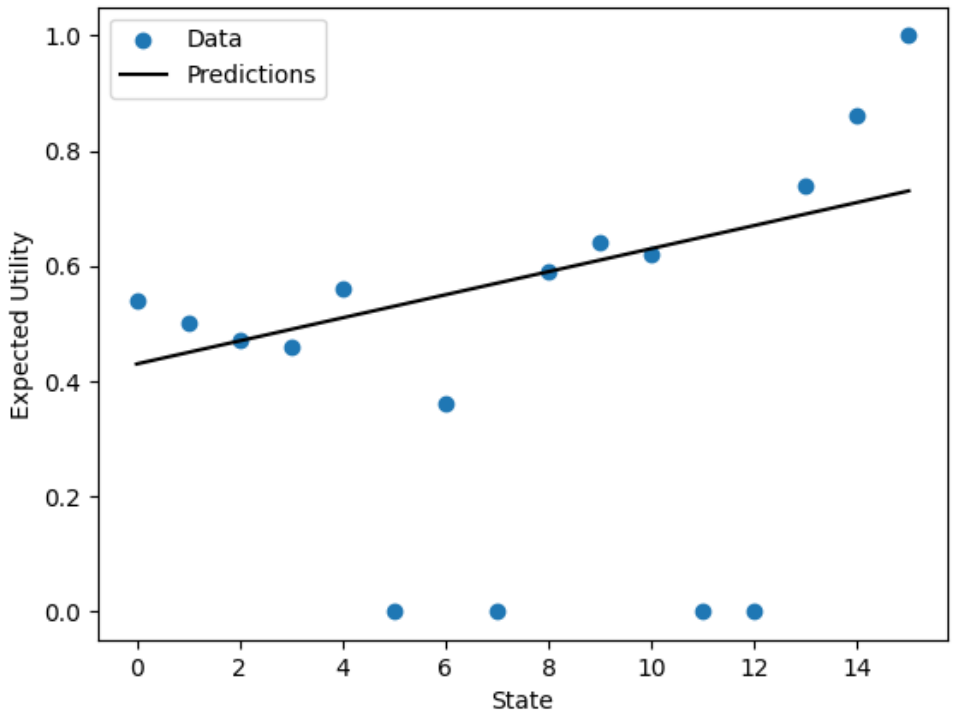
Image by OpenClipArt on FreeSVG


Bias
b
Inputs
x1
x2
xm
ω1
ω2
ωm
Σ
φ(•)
y
Output
Sum
Activation
Function
Weights


b
x
w_1
$\sum$
y
State

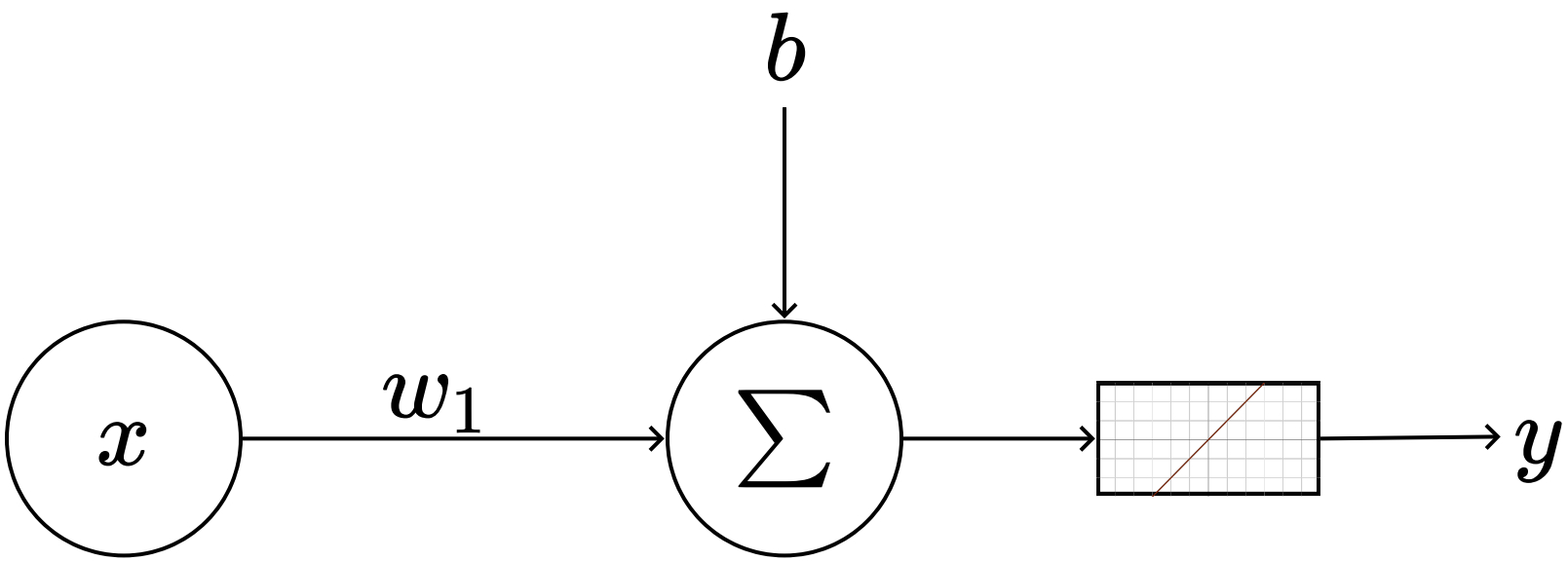

b
x
w_1
$\sum$
y
State
Linear Activation function


b
Utility
x
w_1
$\sum$
y
State
Linear Activation function



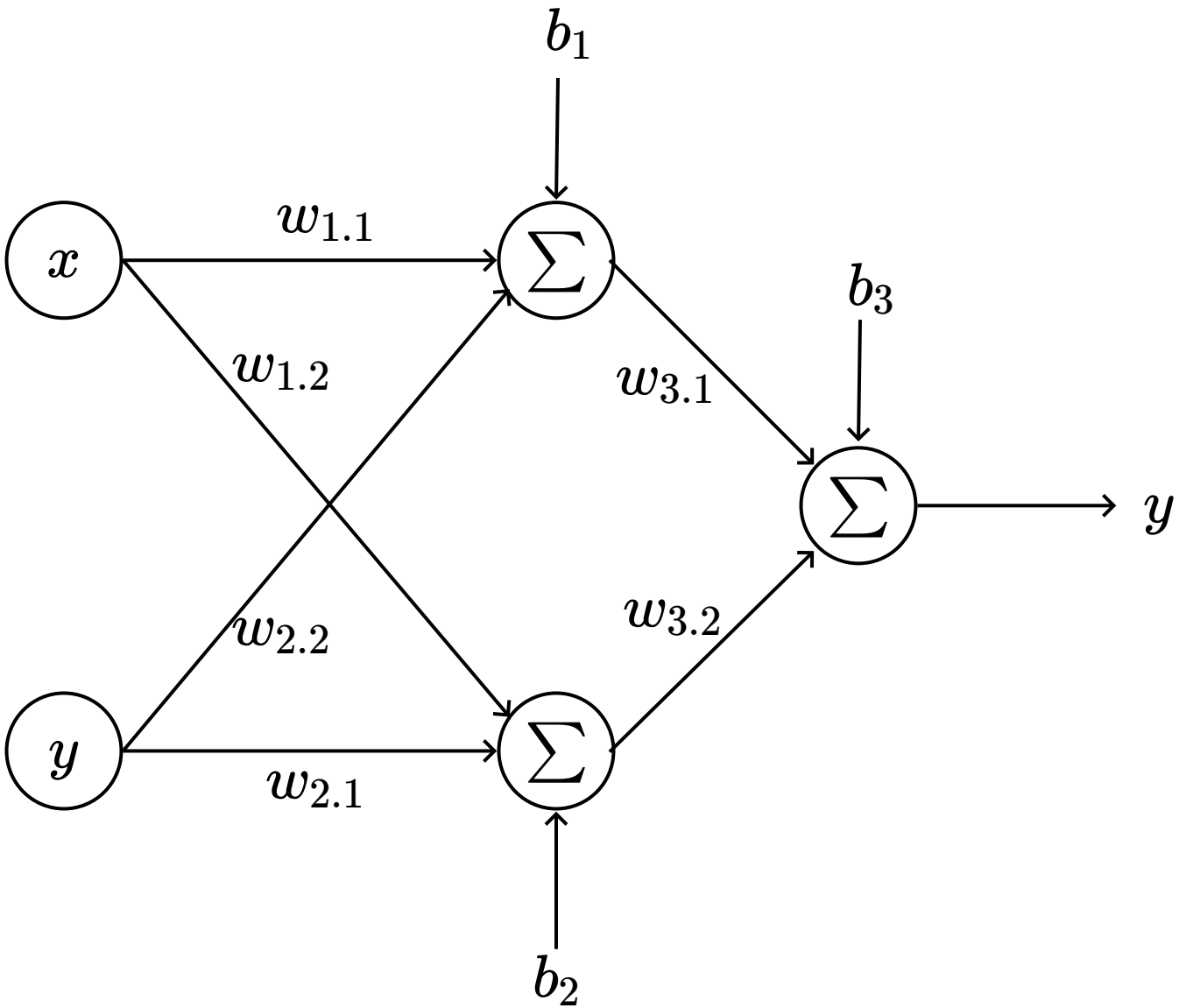
$$\hat{U}_w(x) = w_0 + w_1 x$$

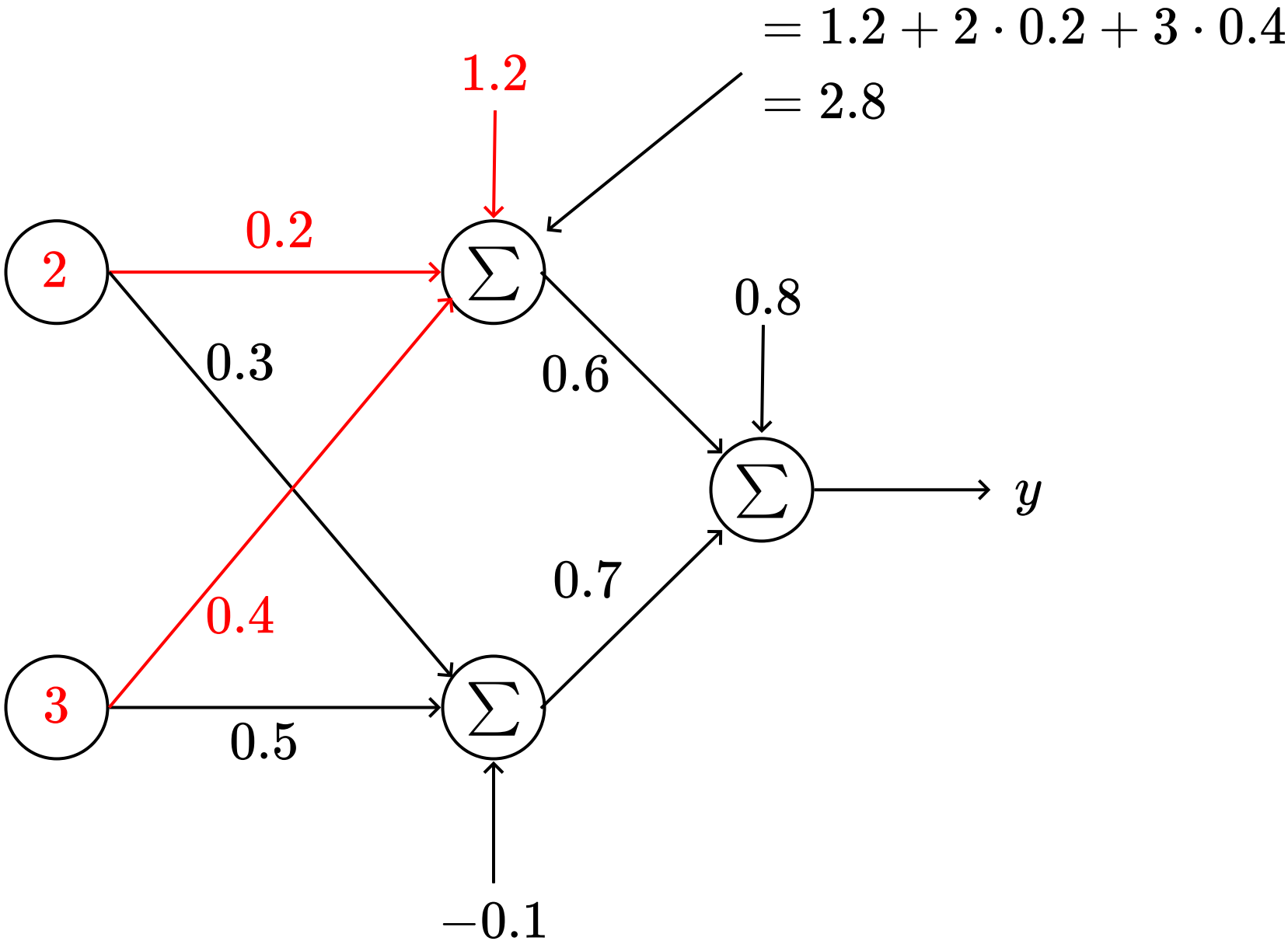
$$\hat{U}_w(x) = 0.43 + 0.02x$$

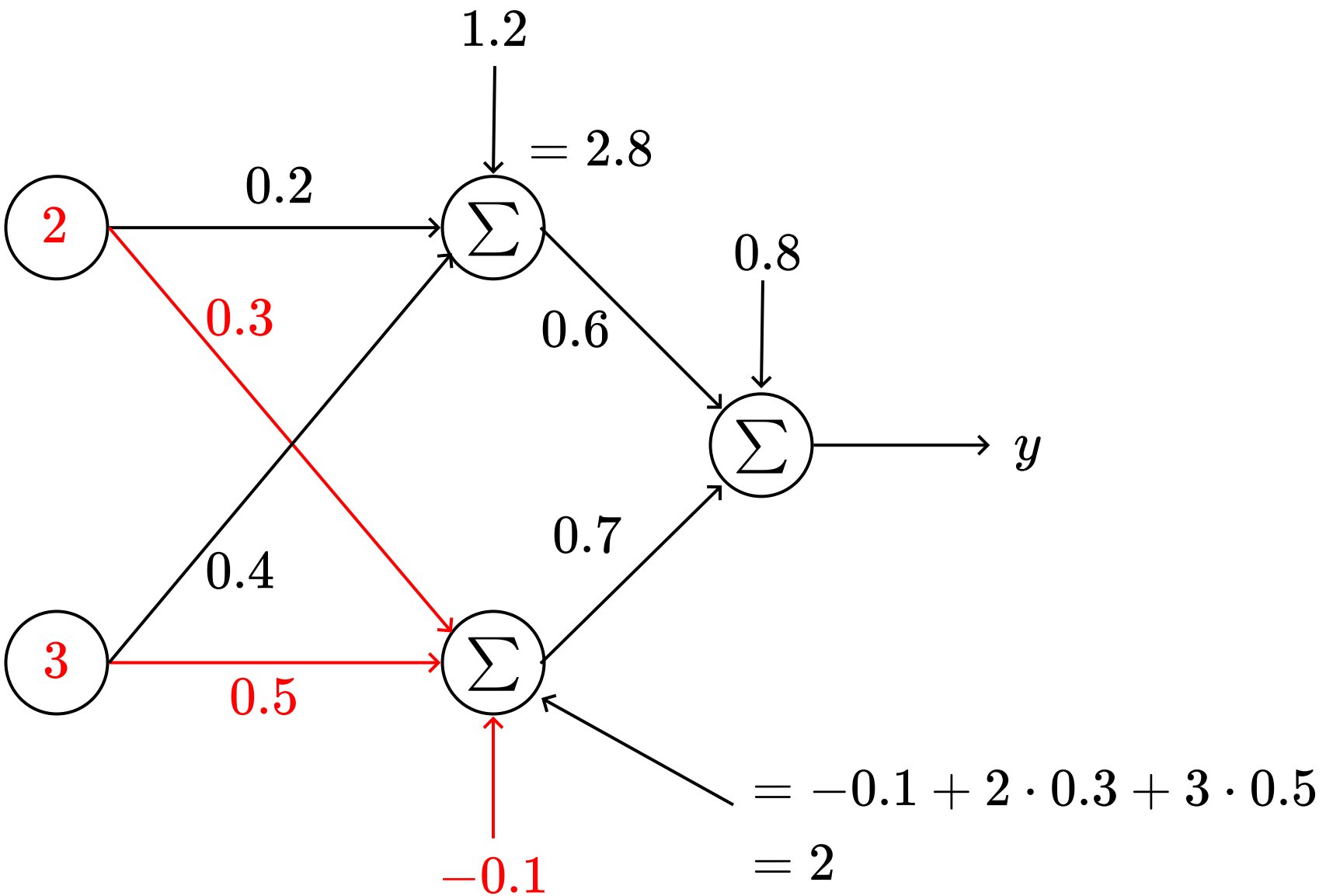


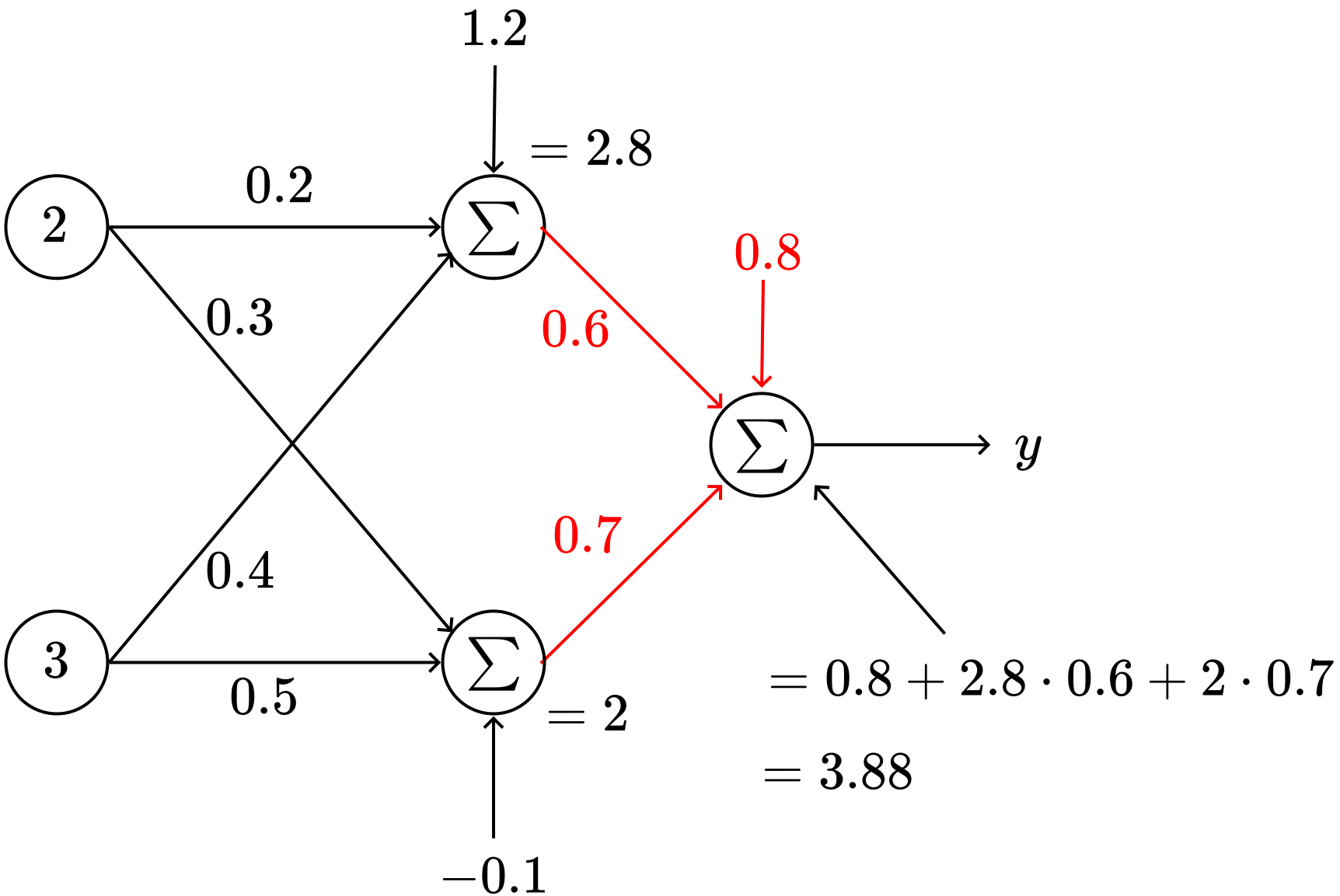
$$\hat{U}_w(x) = w_0 + w_1 x \longrightarrow b = 0.43, w_1 = 0.02$$

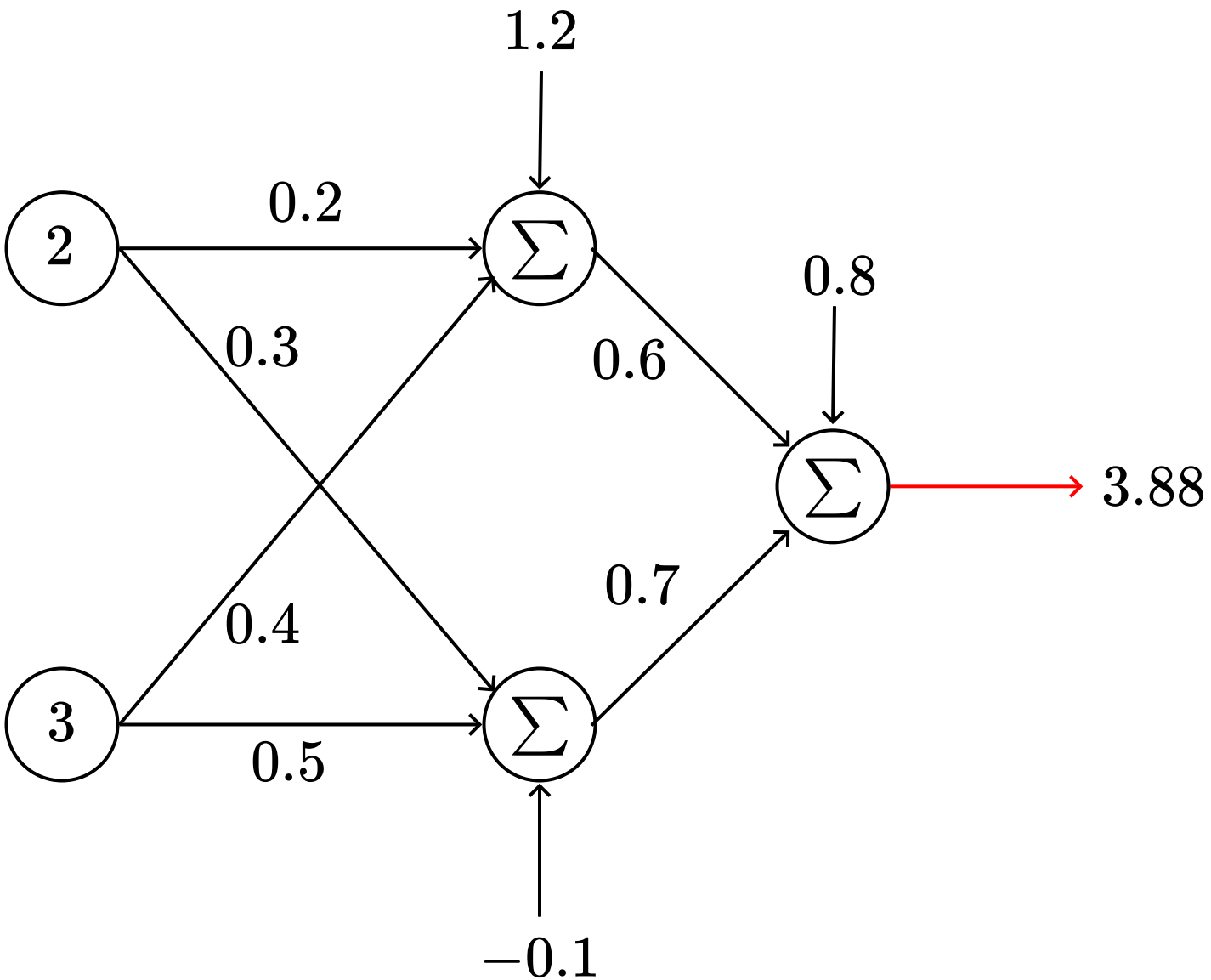
$$\hat{U}_w(x) = 0.43 + 0.02x$$

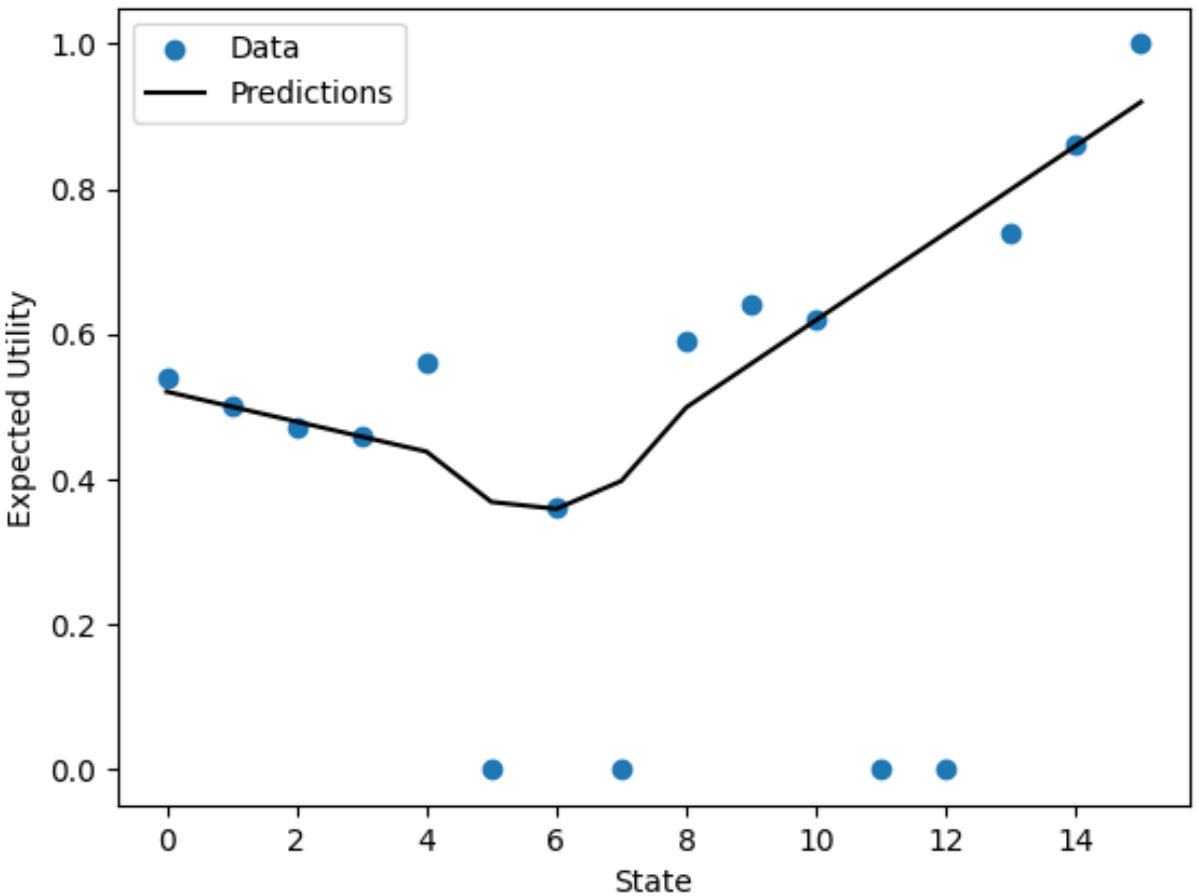

b_1
x
$w_{1.1}$
$\sum$
b_3
$w_{1.2}$
$w_{3.1}$
$\sum$
y
$w_{2.2}$
$w_{3.2}$
y
$\sum$
$w_{2.1}$
b_2

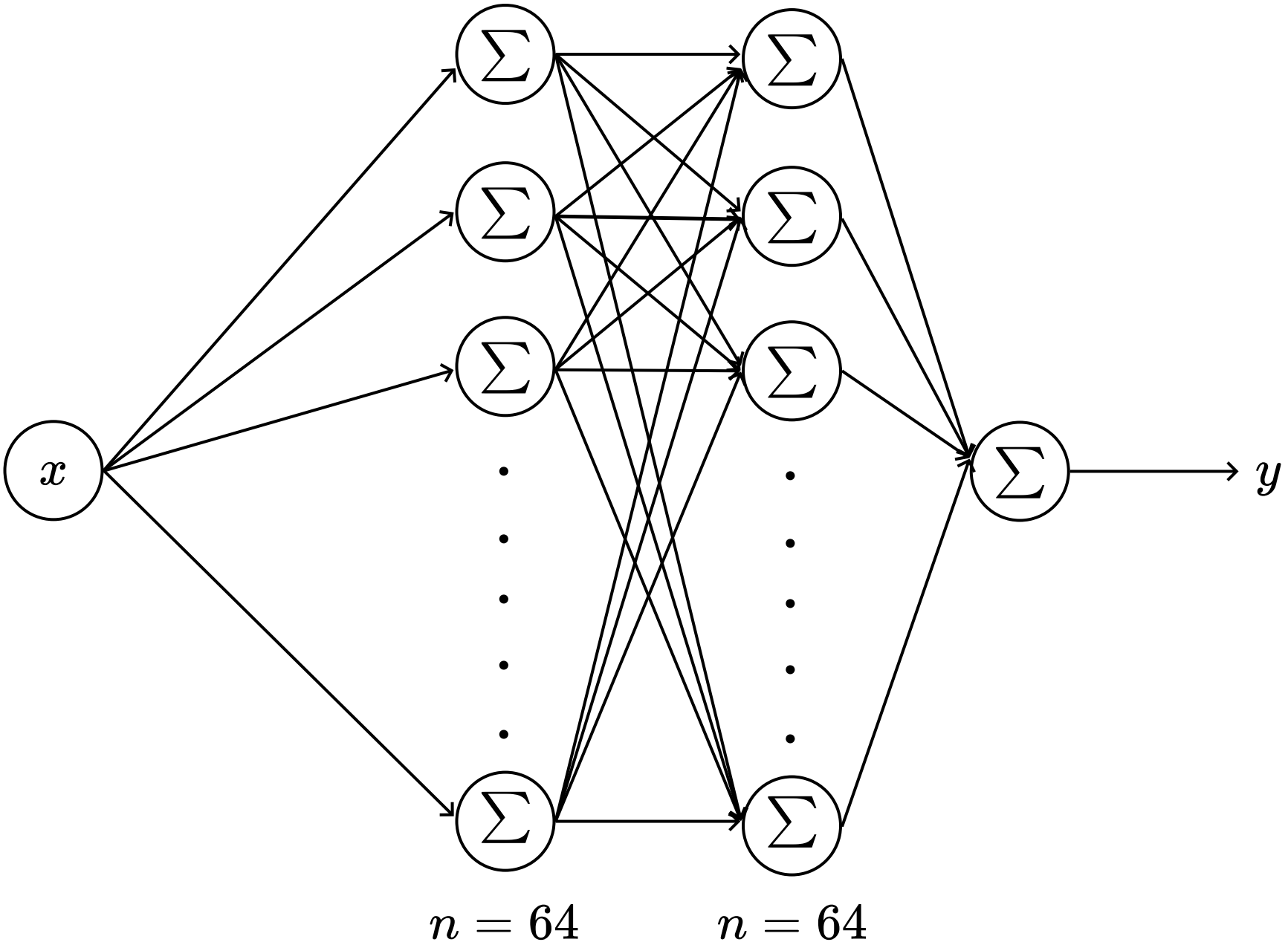

$= 1.2 + 2 \cdot 0.2 + 3 \cdot 0.4$
$= 2.8$
1.2
2
0.2
Σ
0.8
0.3
0.6
Σ
y
0.7
0.4
3
Σ
0.5
-0.1

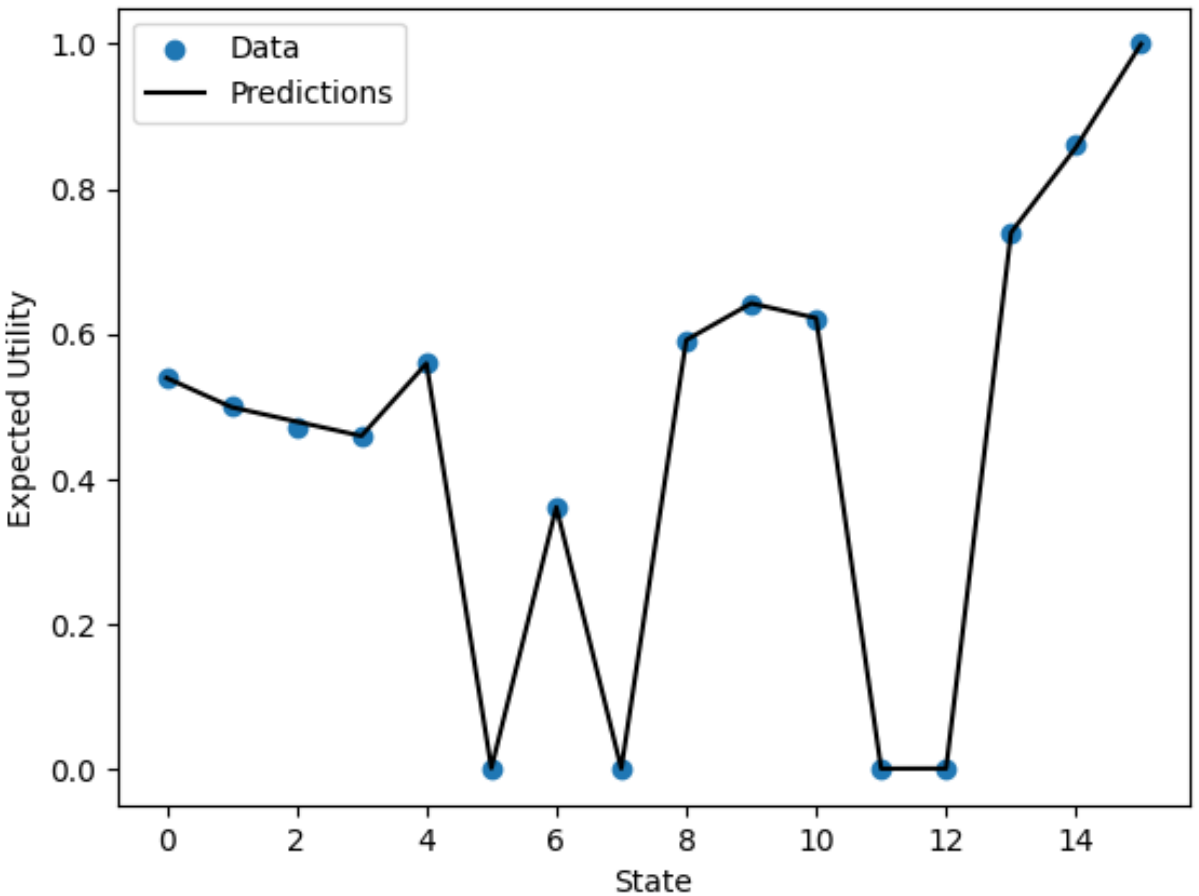

1.2
= 2.8
0.2
2
0.3
0.8
0.6
y
0.7
0.4
3
0.5
= −0.1 + 2 · 0.3 + 3 · 0.5
= 2
−0.1

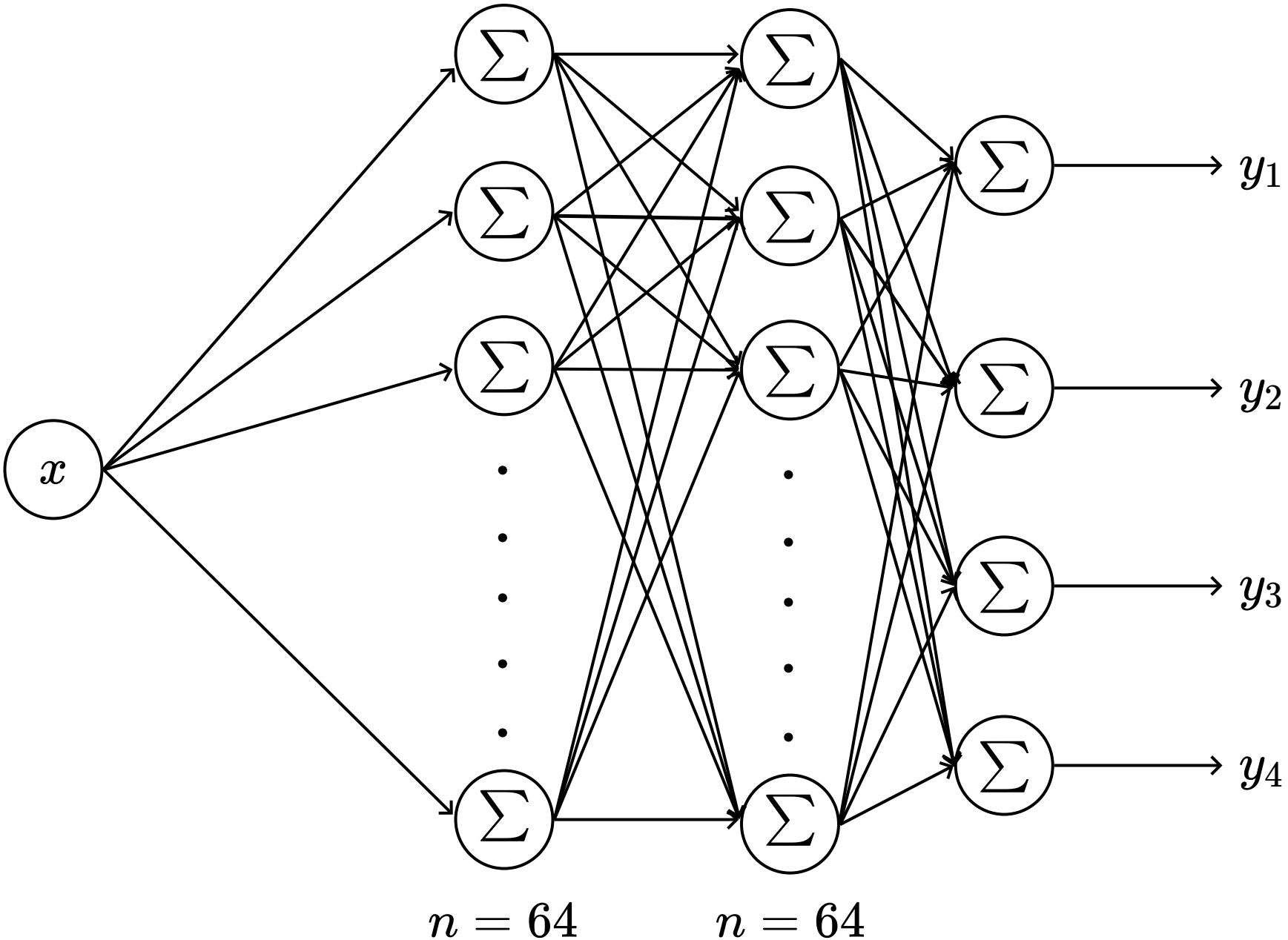

1.2
= 2.8
2
0.2
0.3
0.4
3
0.5
0.6
0.7
0.8
y
= 2
−0.1
= 0.8 + 2.8 · 0.6 + 2 · 0.7
= 3.88

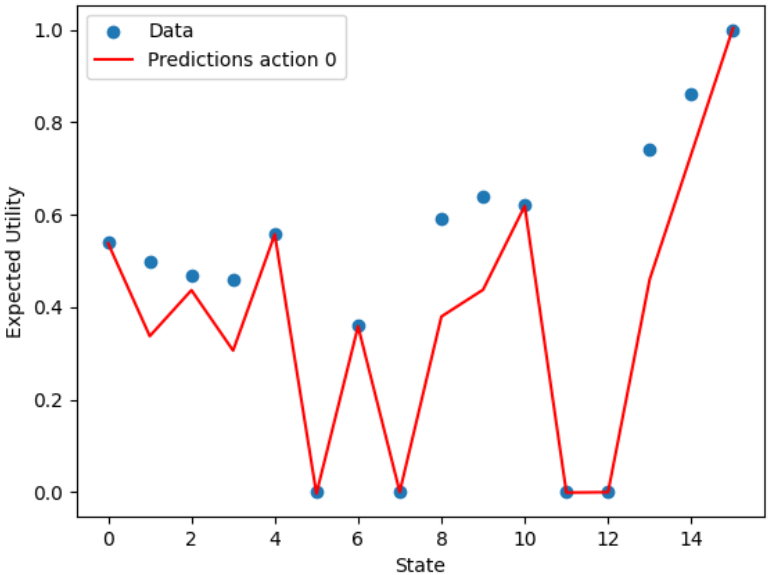

1.2
2
0.2
Σ
0.8
0.3
0.6
Σ
3.88
0.7
0.4
3
Σ
0.5
−0.1

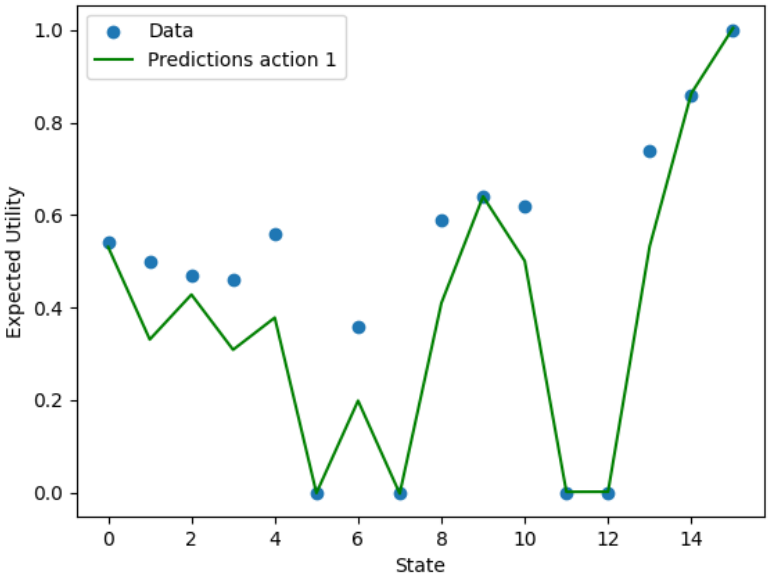

Data
Predictions
Expected Utility
State
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10
12
14

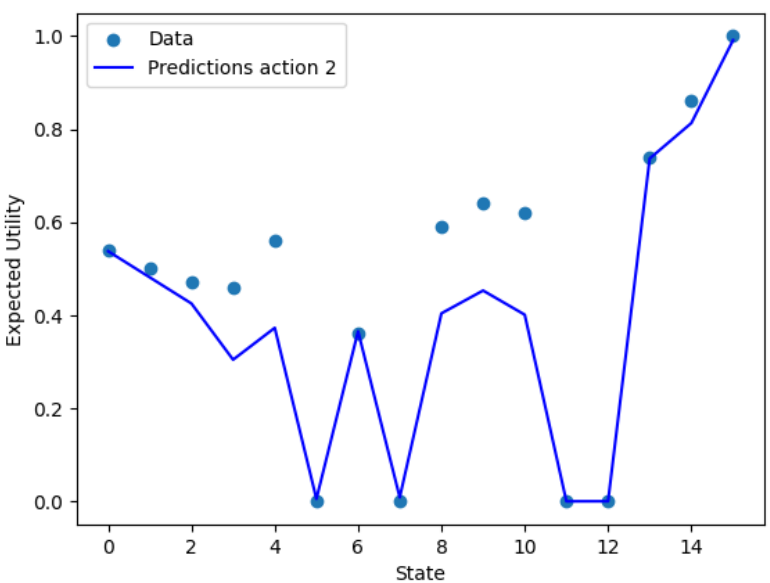

x
Σ
Σ
Σ
Σ
Σ
Σ
Σ
Σ
Σ
y
$n = 64$
$n = 64$

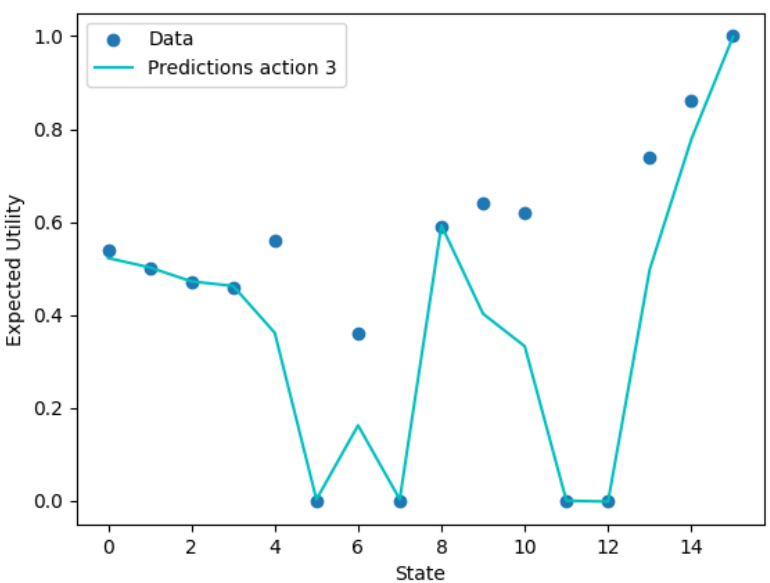

Data
Predictions
Expected Utility
State
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10
12
14


x
Σ
y_1
y_2
y_3
y_4
$n = 64$
$n = 64$


Data
Predictions action 0
Expected Utility
State
0.0
0.2
0.4
0.6
0.8
1.0
0
2
4
6
8
10
12
14


Data
Predictions action 1
Expected Utility
State
0.0
0.2
0.4
0.6
0.8
1.0
0
2
4
6
8
10
12
14


Data
Predictions action 2
Expected Utility
State
0.0
0.2
0.4
0.6
0.8
1.0
0
2
4
6
8
10
12
14


Data
Predictions action 3
Expected Utility
State
0.0
0.2
0.4
0.6
0.8
1.0
0
2
4
6
8
10
12
14

- Recap
- Generalization in Reinforcement Learning
- Artificial Neural Networks
- Bellman Update for ANNs
- Deep Q-Network

$$Q(s,a) \leftarrow Q(s,a) + \alpha[R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$

$$w \leftarrow w + \alpha[(R(s,a,s') + \gamma \max_{a'} Q(s',a',w) - Q(s,a,w))\nabla_w Q(s,a,w)]$$

$$Q(s,a) \leftarrow Q(s,a) + \alpha[R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$

$$w \leftarrow w + \alpha[(R(s,a,s') + \gamma \max_{a'} Q(s',a',w) - Q(s,a,w))\nabla_w Q(s,a,w)]$$

$$Q(s,a) \leftarrow Q(s,a) + \alpha[R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$

$$w \leftarrow w + \alpha[(R(s,a,s') + \gamma \max_{a'} Q(s',a',w) - Q(s,a,w))\nabla_w Q(s,a,w)]$$

$$Q(s,a) \leftarrow Q(s,a) + \alpha[R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$

$$w \leftarrow w + \alpha[(R(s,a,s') + \gamma \max_{a'} Q(s',a',w) - Q(s,a,w))\nabla_w Q(s,a,w)]$$

- Recap
- Generalization in Reinforcement Learning
- Artificial Neural Networks
- Bellman Update for ANNs
- Deep Q-Network

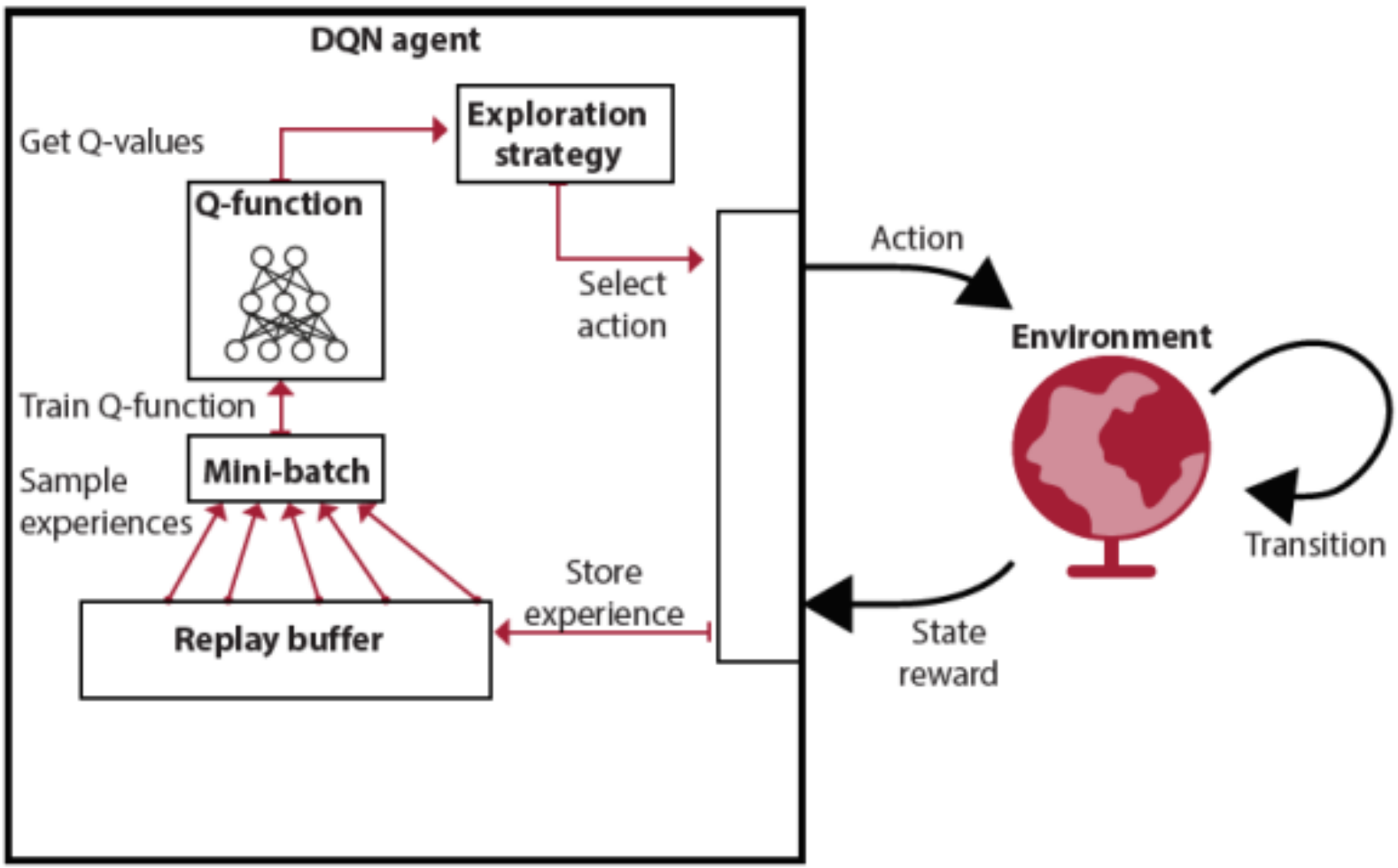


Image from Grokking Deep Reinforcement Learning

Experience Replay

Incremental methods have several problems:

- They are not sample-efficient.
- They do not work well with popular (and computationally efficient) ML algorithms.
- They may forget rare experiences that would be useful later on (catastrophic forgetting).

Experience replay and prioritized experience replay address these issues by storing experiences and reusing them. The experiences can then be sampled uniformly or by prioritization.

Deep Q-Network

```
initialize replay memory D
initialize action-value function Q with random weights
observe initial state s
repeat
    select an action a
        with probability ε select a random action
        otherwise select a = argmaxa'Q(s,a')
    carry out action a
    observe reward r and new state s'
    store experience <s, a, r, s'> in replay memory D

    sample random transitions <ss, aa, rr, ss'> from replay memory D
    calculate target for each minibatch transition
        if ss' is terminal state then tt = rr
        otherwise tt = rr + γmaxa'Q(ss', aa')
    train the Q network using (tt - Q(ss, aa))^2 as loss

    s = s'
until terminated
```

- Recap
- Generalization in Reinforcement Learning
- Artificial Neural Networks
- Bellman Update for ANNs
- Deep Q-Network

Questions?

$$Q(s,a) \leftarrow Q(s,a) + \alpha[R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$

$$\downarrow$$

$$w \leftarrow w + \alpha[(R(s,a,s') + \gamma \max_{a'} Q(s',a',w) - Q(s,a,w))\nabla_w Q(s,a,w)]$$

$$w \leftarrow w - \tfrac{1}{2}\alpha\nabla_w[(R(s,a,s') + \gamma \max_{a'} Q(s',a',w) - Q(s,a,w))^2]$$

$$\nabla_{\theta_i} L_i(\theta_i) \leftarrow \mathbb{E}_{s,a,r,s'}[(r + \gamma \max_{a'} Q(s',a';\theta_i) - Q(s,a;\theta_i))\nabla_{\theta_i} Q(s,a;\theta_i)]$$

$$\Delta_w = -\tfrac{1}{2}\alpha\nabla_w[(R(s,a,s') + \gamma \max_{a'} Q(s',a',w) - Q(s,a,w))^2]$$

$$\Delta_w = \alpha[(R(s,a,s') + \gamma \max_{a'} Q(s',a',w) - Q(s,a,w))\nabla_w Q(s,a,w)]$$