Slide 1 — Reinforcement Learning: Levels 1–3

Goal
Understand how RL grows from simple tables to neural networks.

Levels: - Level 1: Tabular Q-learning - Level 2: Q-learning with tricks (replay, stability) - Level 3: Deep Q-Network (DQN)

Slide 2 — Common Setup (All Levels)

Environment - Provides state, reward, next state

Agent - Chooses actions

Loop 1. Observe state 2. Take action 3. Get reward + next state 4. Learn

TH
โครงสร้างนี้เหมือนกันทุกระดับ

Slide 3 — Level 1: Tabular Q-Learning (Idea)

State-action table (Q-table) - Rows = states - Columns = actions

Learning: - Update table entry directly

Works when: - States are small and discrete

Slide 4 — Level 1: Simple Example

State: traffic light - red, yellow, green

Actions: - stop, go

Q-table stores: - How good each action is in each state

Human analogy
Memorizing rules from experience

Slide 5 — Why Level 1 Is Limited

Problems: - Too many states → huge table - Continuous values impossible

CartPole has infinite states → table breaks

Slide 6 — Level 2: Toward Generalization

Key ideas added: - Same Q-learning update - But focus on **stability and reuse of experience**

Concepts introduced: - Replay memory (idea) - Delayed updates

Still mostly conceptual

Slide 7 — Level 2: Human Learning Analogy

Instead of: - Forgetting after each step

We: - Remember past experiences - Review them multiple times

Like: - Practicing old exam questions

Slide 8 — Why We Need Level 3

Reality: - State space is continuous - Table is impossible

Solution:

> Replace Q-table with a neural network

Slide 9 — Level 3: Deep Q-Network (DQN)

What changes - Q(s, a) ≈ Neural Network(s)

What stays - Q-learning idea - Bellman equation

Slide 10 — DQN Architecture

Input: - State (4 values in CartPole)

Hidden layer: - 128 neurons - Distributed representation

Output: - Q-value for each action

Slide 11 — Why Neural Networks Help

They: - Generalize across states - Learn useful combinations - Handle continuous inputs

Human analogy
Recognizing patterns, not memorizing cases

Slide 12 — Replay Buffer (Memory)

Stores: - state - action - reward - next_state - done

Why: - Break correlation - Learn from the past

Slide 13 — Target Network (Stability)

Problem: - Network learning from itself

Solution: - Keep a delayed copy - Update it slowly

Target ≠ goal
Target = stable reference value

Slide 14 — Forward vs Backward (Level 3)

Forward pass - Decide action

Loss computation - Compare predicted Q vs target Q

Backward pass - Update weights

Slide 15 — Full DQN Code (VS Code Ready)

Below is a complete minimal example.

Slide 16 — Full Code: Imports

```
import gymnasium as gym
import random
from collections import deque
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
```

Slide 17 — Q-Network

```
class QNetwork(nn.Module):
    def __init__(self, state_dim, action_dim):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(state_dim, 128),
            nn.ReLU(),
            nn.Linear(128, action_dim)
        )

    def forward(self, x):
        return self.net(x)
```

Slide 18 — Replay Buffer

```
class ReplayBuffer:
    def __init__(self, capacity=10000):
        self.buffer = deque(maxlen=capacity)

    def push(self, s, a, r, ns, d):
```

```
        self.buffer.append((s, a, r, ns, d))

    def sample(self, batch_size):
        batch = random.sample(self.buffer, batch_size)
        s, a, r, ns, d = zip(*batch)
        return (
            torch.tensor(s, dtype=torch.float32),
            torch.tensor(a, dtype=torch.int64),
            torch.tensor(r, dtype=torch.float32),
            torch.tensor(ns, dtype=torch.float32),
            torch.tensor(d, dtype=torch.float32),
        )

    def __len__(self):
        return len(self.buffer)
```

Slide 19 — Training Step

```
def train_step():
    if len(buffer) < BATCH_SIZE:
        return

    s, a, r, ns, d = buffer.sample(BATCH_SIZE)

    q = policy_net(s).gather(1, a.unsqueeze(1)).squeeze(1)

    with torch.no_grad():
        target = r + GAMMA * target_net(ns).max(1)[0] * (1 - d)

    loss = nn.MSELoss()(q, target)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

Slide 20 — Main Loop

```
for episode in range(500):
    state, _ = env.reset()
    total_reward = 0
```

```
    while True:
        action = select_action(state)
        next_state, reward, done, _, _ = env.step(action)
        buffer.push(state, action, reward, next_state, done)
        state = next_state
        total_reward += reward
        train_step()
        if done:
            break
```

Slide 21 — Summary of Levels

Level	Representation	Key Idea
1	Q-table	Memorization
2	Q-learning + memory	Stability
3	Neural network	Generalization

Slide 22 — Final Takeaway

RL evolves from memorizing values to learning representations.
