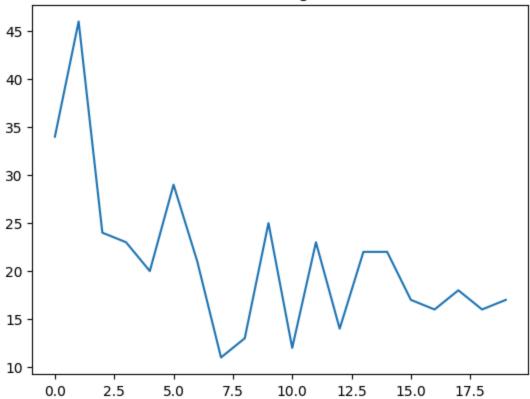
```
In [4]: import gymnasium as gym
        import numpy as np
        import torch
        import torch.nn as nn
        import torch.optim as optim
        from collections import deque
        import matplotlib.pyplot as plt
        # Hyperparameters (optimized for speed)
        EPISODES = 20
        BATCH SIZE = 32
        GAMMA = 0.95
        LEARNING RATE = 0.001
        MEMORY SIZE = 1000
        EPSILON DECAY = 0.97
        # Environment setup
        env = gym.make('CartPole-v1')
        state size = env.observation space.shape[0]
        action size = env.action space.n
        # Simplified network
        class DQN(nn.Module):
            def __init__(self):
                super(). init ()
                self.net = nn.Sequential(
                    nn.Linear(state size, 16),
                    nn.ReLU(),
                    nn.Linear(16, action size)
            def forward(self, x):
                return self.net(x)
        # Initialize components
        policy net = DQN()
        target net = DQN()
        target net.load state dict(policy net.state dict())
        optimizer = optim.Adam(policy net.parameters(), lr=LEARNING RATE)
        memory = deque(maxlen=MEMORY SIZE)
        epsilon = 1.0
        def train():
            if len(memory) < BATCH SIZE:</pre>
                return
            # Sample batch and fix dimensions
            batch = random.sample(memory, BATCH SIZE)
            states = torch.FloatTensor(np.array([t[0] for t in batch])) # Shape: [E
            actions = torch.LongTensor([t[1] for t in batch])
            rewards = torch.FloatTensor([t[2] for t in batch])
            next states = torch.FloatTensor(np.array([t[3] for t in batch]))
            dones = torch.FloatTensor([t[4] for t in batch])
            # 0-value calculation
```

```
current q = policy net(states).gather(1, actions.unsqueeze(-1))
   next q = target net(next states).max(1)[0].detach()
   expected q = rewards + (1 - dones) * GAMMA * next q
   # Update model
   loss = nn.MSELoss()(current q.squeeze(), expected q)
   optimizer.zero grad()
   loss.backward()
   optimizer.step()
# Training loop
scores = []
for episode in range(EPISODES):
   state, _ = env.reset()
   score = 0
   for _ in range(200): # Max steps
        # Epsilon-greedy action
       if np.random.rand() < epsilon:</pre>
           action = env.action space.sample()
       else:
           with torch.no grad():
                action = policy net(torch.FloatTensor(state)).argmax().item(
       # Environment step
       next state, reward, terminated, truncated, = env.step(action)
       done = terminated or truncated
       # Store experience (flatten state)
       memory.append((state, action, reward, next state, done))
        state = next state
       score += reward
       # Train
       train()
       if done:
           break
   # Update parameters
   epsilon *= EPSILON DECAY
   target net.load state dict(policy net.state dict())
   scores.append(score)
   print(f"Ep {episode+1}/{EPISODES}, Score: {score}, \(\epsilon:.2f\)")
# Quick plot
plt.plot(scores)
plt.title('DQN Training (Fast)')
plt.show()
```

```
Ep 1/20, Score: 34.0, ε: 0.97
Ep 2/20, Score: 46.0, ε: 0.94
Ep 3/20, Score: 24.0, ε: 0.91
Ep 4/20, Score: 23.0, ε: 0.89
Ep 5/20, Score: 20.0, ε: 0.86
Ep 6/20, Score: 29.0, ε: 0.83
Ep 7/20, Score: 21.0, ε: 0.81
Ep 8/20, Score: 11.0, ε: 0.78
Ep 9/20, Score: 13.0, ε: 0.76
Ep 10/20, Score: 25.0, ε: 0.74
Ep 11/20, Score: 12.0, ε: 0.72
Ep 12/20, Score: 23.0, ε: 0.69
Ep 13/20, Score: 14.0, ε: 0.67
Ep 14/20, Score: 22.0, ε: 0.65
Ep 15/20, Score: 22.0, ε: 0.63
Ep 16/20, Score: 17.0, ε: 0.61
Ep 17/20, Score: 16.0, ε: 0.60
Ep 18/20, Score: 18.0, ε: 0.58
Ep 19/20, Score: 16.0, ε: 0.56
Ep 20/20, Score: 17.0, ε: 0.54
```

DQN Training (Fast)



This notebook was converted with convert.ploomber.io