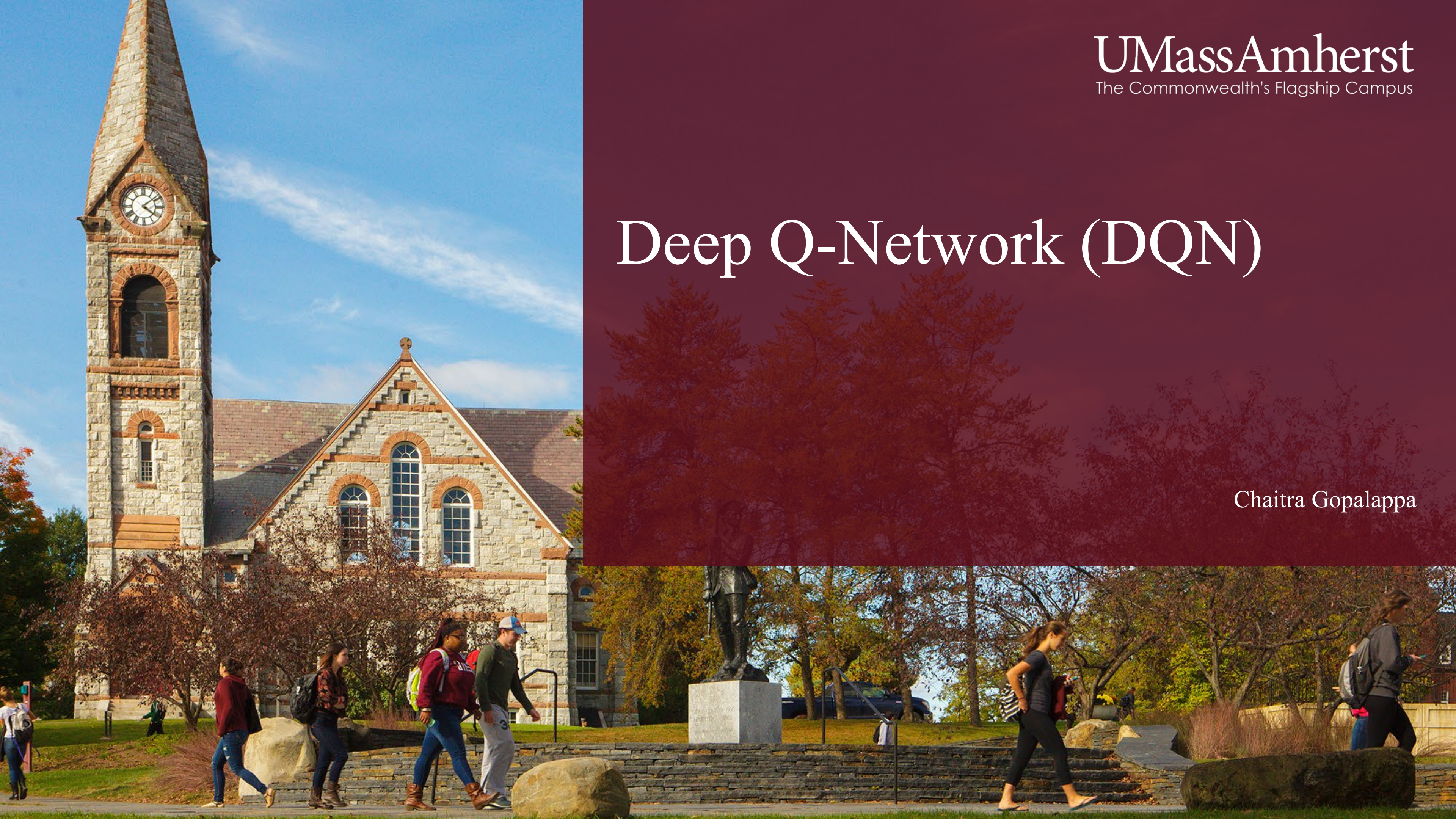


UMassAmherst
The Commonwealth's Flagship Campus

Deep Q-Network (DQN)

Chaitra Gopalappa

Reference

- Chapters 9, 10, 11 of Sutton and Barto
  - Provide general understanding of function approximation
- Practical advances over past decade
  - Research articles best source
- Chapters 6 and 7, Lapan
  - Focusses on problems needing high-computational resources (1-2 days to converge)
  - Review it for computational efficiency
    - Use of wrappers to simplify code writing
    - Use of PTAN libraries for simplifying functionality
  - For this class: I will instead use simple examples (computationally doable)

Notations

- Random variables in Capital
- Vector in bold small

$\boldsymbol{s}$: *state vector*

$$\boldsymbol{s} = [S_1, \dots S_N]$$

$\boldsymbol{a}$: *action vector*

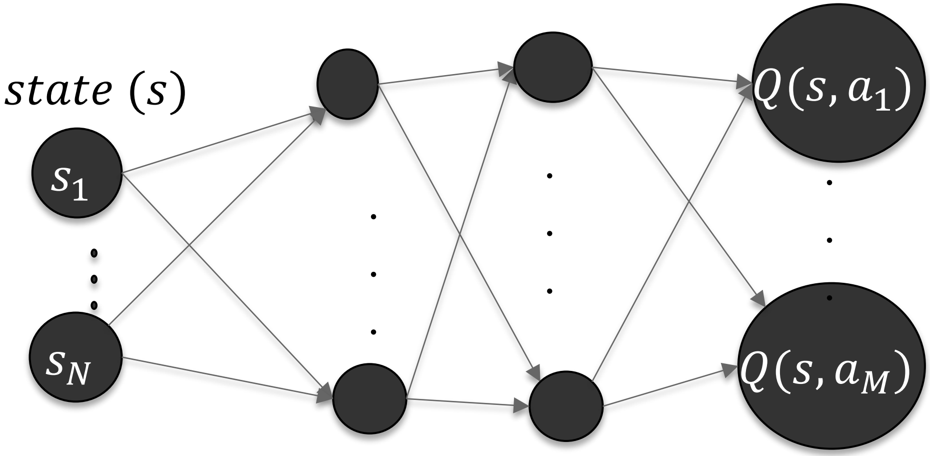
$$\boldsymbol{a} = [A_1, \dots A_M]$$

Deep-Q Network (DQN)

- Work over past decade

DQN architecture

- Learns Q-values
- Actions in the output layer
- Size of output layer = number of actions



Deep-Q Network

Mnih, V. , **Playing Atari with Deep Reinforcement Learning, 2013,** https://doi.org/10.48550/arXiv.1312.5602

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation $\boxed{3}$
  end for
end for

- New feature: Experience replay:
  - Agent selects and executes an action according to an epsilon-greedy policy
  - store the agent's experiences at each time-step, $e_t = (s_t, a_t, r_t, s_{t+1})$ in a data-set $D = e_1, \ldots, e_N$, pooled over many episodes into a replay memory
  - Randomly draw minibatch size number of samples from D

Deep-Q Network

Mnih, V. , **Playing Atari with Deep Reinforcement Learning, 2013,** https://doi.org/10.48550/arXiv.1312.5602

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation $\boxed{3}$
  end for
end for

- Uses same NN for updating the transition state as well.

Deep-Q Network

Mnih, V. , **Playing Atari with Deep Reinforcement Learning, 2013,** https://doi.org/10.48550/arXiv.1312.5602

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

- Mnih, V. et. al., Human-level control through deep reinforcement learning, Nature, 2015 https://www.nature.com/articles/nature14236

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
For episode $= 1, M$ **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t = 1$,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

WITH TARGET NETWORK (use different NN for updating transition state)

Every C steps reset the target network to actual network

Note: ϕ are transformation of states, e.g., in games multiple snapshots of an image can be represented as a 'state'

Deep-Q Network

Mnih, V. , **Playing Atari with Deep Reinforcement Learning, 2013,** https://doi.org/10.48550/arXiv.1312.5602

- Mnih, V. et. al., Human-level control through deep reinforcement learning, Nature, 2015 https://www.nature.com/articles/nature14236

Note: ϕ are transformations done on raw data to convert them to a standardized ‘state’, e.g., in games multiple snapshots of an image can be represented as a ‘state’

In examples in this class, we ignore this

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
For episode $= 1, M$ **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t = 1$,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \mathrm{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

DQN algorithm
(without the state feature transformation)

Initialize $Q(s,a)$ with random weights θ
Initialize target function $\hat{Q}(s,a)$ with random weights $\hat{\theta}$
Set $\varepsilon \leftarrow 1.0$
Initialize replay buffer memory D to capacity N
For episode = 1 to M, do
    For t=1 to T (end of episode) do

1. With probability ε , select a random action, a_t; otherwise, $a_t = argmax_a Q(s,a)$
2. Execute action a_t in an emulator and observe the reward,r_t, and the next state, $s_{t+1}{}'$.
3. Store transition $(s_t, a_t, r_t, s_{t+1}{}')$ in the replay buffer D.
4. Sample a random mini-batch of transitions $(s_j, a_j, r_j, s'_{j+1})$ in from the replay buffer D.
5. For every transition, calculate target
    1. $$y_j = \begin{cases} r_j \; if \; episode \; has \; terminated \; in \; j+1 \\ r_j + \gamma \max_{a' \in A} \hat{Q}(s_{j+1}{}', a'), otherwise \end{cases}$$
6. Calculate loss: $\mathcal{L} = \left(Q\left(s_j, a_j; \theta\right) - y_j\right)^2$ and update $Q(s,a;\theta)$ using the SGD algorithm by minimizing the loss with respect to network parameters θ.
7. Every C steps, set $\hat{Q} = Q$ (copy weights from Q to $\hat{Q}$).

    End t loop
End episode loop

UMassAmherst

The Commonwealth's Flagship Campus