Lecture 7:
Function Approximation, SGD, DQN

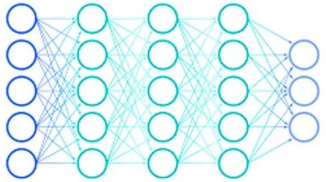
B. Ravindran

Need for Function Approximation

- Issues with large state/action spaces:
  - tabular approaches not memory efficient
  - data sparsity
  - continuous state/action spaces
  - generalization
- Use a parameterized representation
  - Value Functions
  - Policies
  - Models

Non-Linear Function Approximator

- ❑ Linear function approximators are restrictive. Can only model linear functions
- ❑ Basis expansion does help to generate non-linear functions in the original input space



- ❑ Non-linear approximators can model complex functions and are very powerful
- ❑ The features are learnt on the fly and are not hard coded as is the case with tile and sparse coding
- ❑ Can generalize to unseen states
- ❑ Disadvantage: Requires a lot of data and compute

Gradient Descent

- Gradient Descent- first-order iterative optimization algorithm for finding a local minimum of a differentiable function

- Compute the gradient of the function at the current point and take a step in the opposite direction. This is the direction of steepest descent

- The logic behind Gradient Descent can be understood by considering the Taylor Series expansion of the function around the current point, up to first order

$$f(x_0 + h) \approx f(x_0) + h \cdot f'(x_0)$$

$$L(\theta + \alpha\Delta\theta) \approx L(\theta) + \alpha\Delta\theta^T \nabla_\theta L(\theta)$$

Semi Gradient Methods

- While computing the gradient of the TD error in Q-learning, we typically ignore the gradient of the TD target
- Hence, it is a Semi Gradient method i.e we are computing an approximation of the true gradient

$$Q(s_t, a_t) = \phi^T(s_t, a_t) \times w_t$$

$$\delta_t = r_{t+1} + \gamma \max_a Q(s_t, a) - Q(s_t, a_t)$$ TD-ERROR

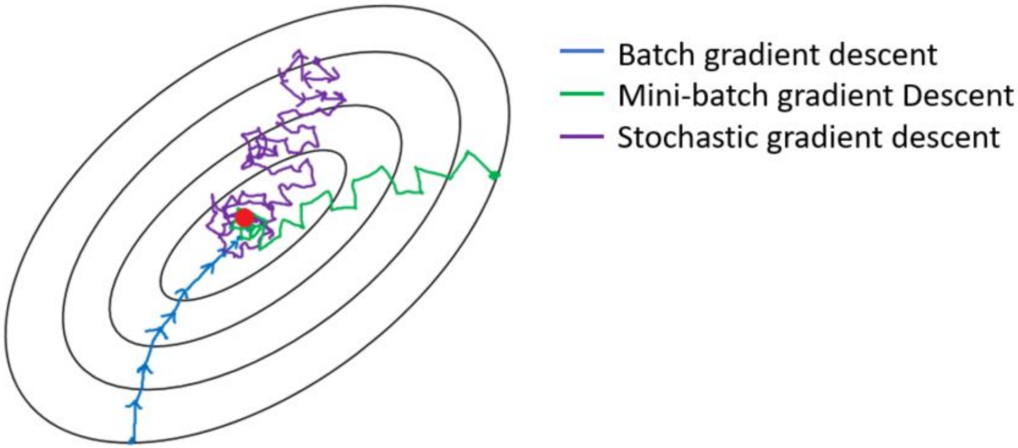
$$\nabla_{w_t} \left[r_{t+1} + \gamma \max_a Q(s_t, a) - Q(s_t, a_t) \right]^2 = -\delta_t \phi(s_t, a_t)$$

$$w_{t+1} = w_t + \alpha \delta_t \phi(s_t, a_t)$$

Stochastic Gradient Descent

- In Stochastic Gradient Descent, the true gradient of the loss function is approximated by the gradient at a single example.

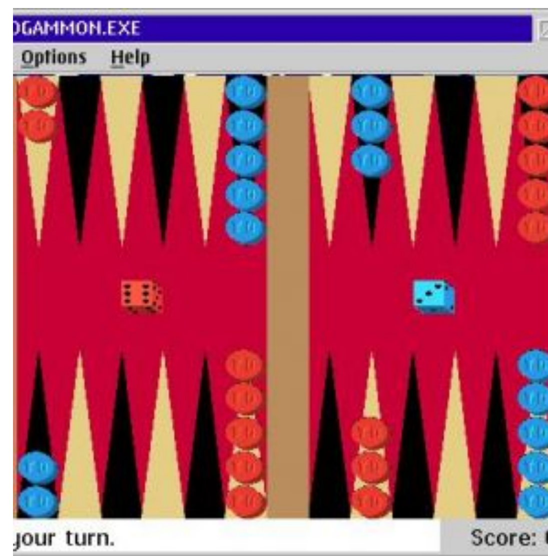
- In practice, we usually perform Mini Batch Gradient Descent, where we compute the gradient using a mini batch of examples. This allows for more efficient computation and smoother convergence.



TD-Gammon

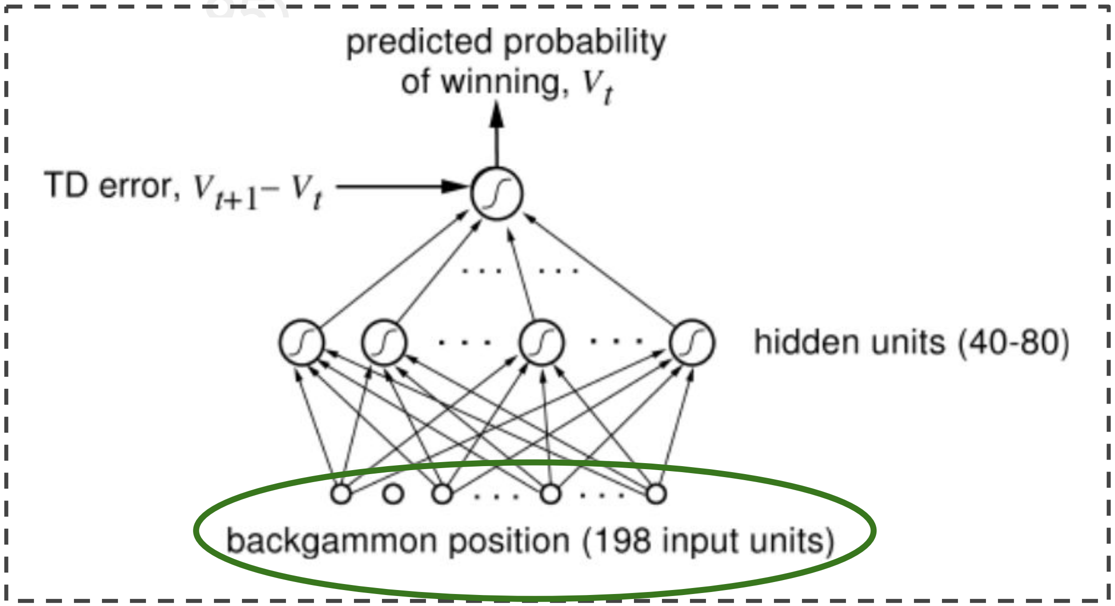
- TD-Gammon (Tesauro 92, 94, 95)
- Beat the best human player in 1995

- Learnt completely by self play
- New moves not recorded by humans in centuries of play

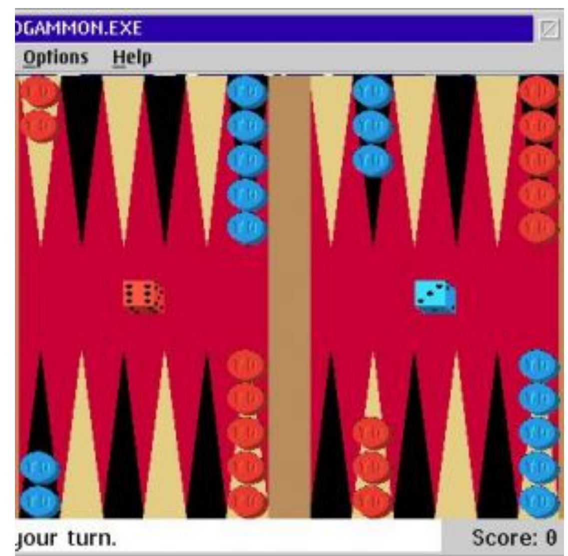


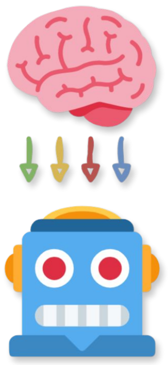
TD-Gammon

❑ TD-Gammon (Tesauro 92, 94, 95)



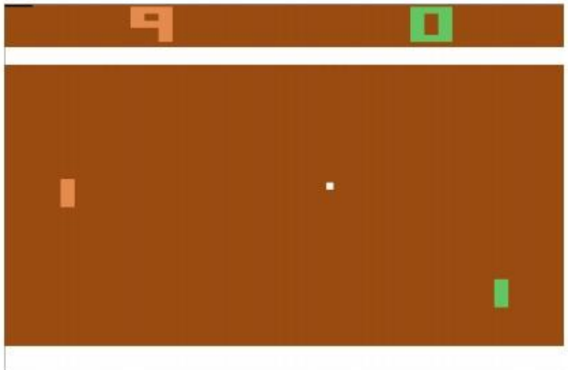
❑ New moves not recorded by humans in centuries of play





DQN

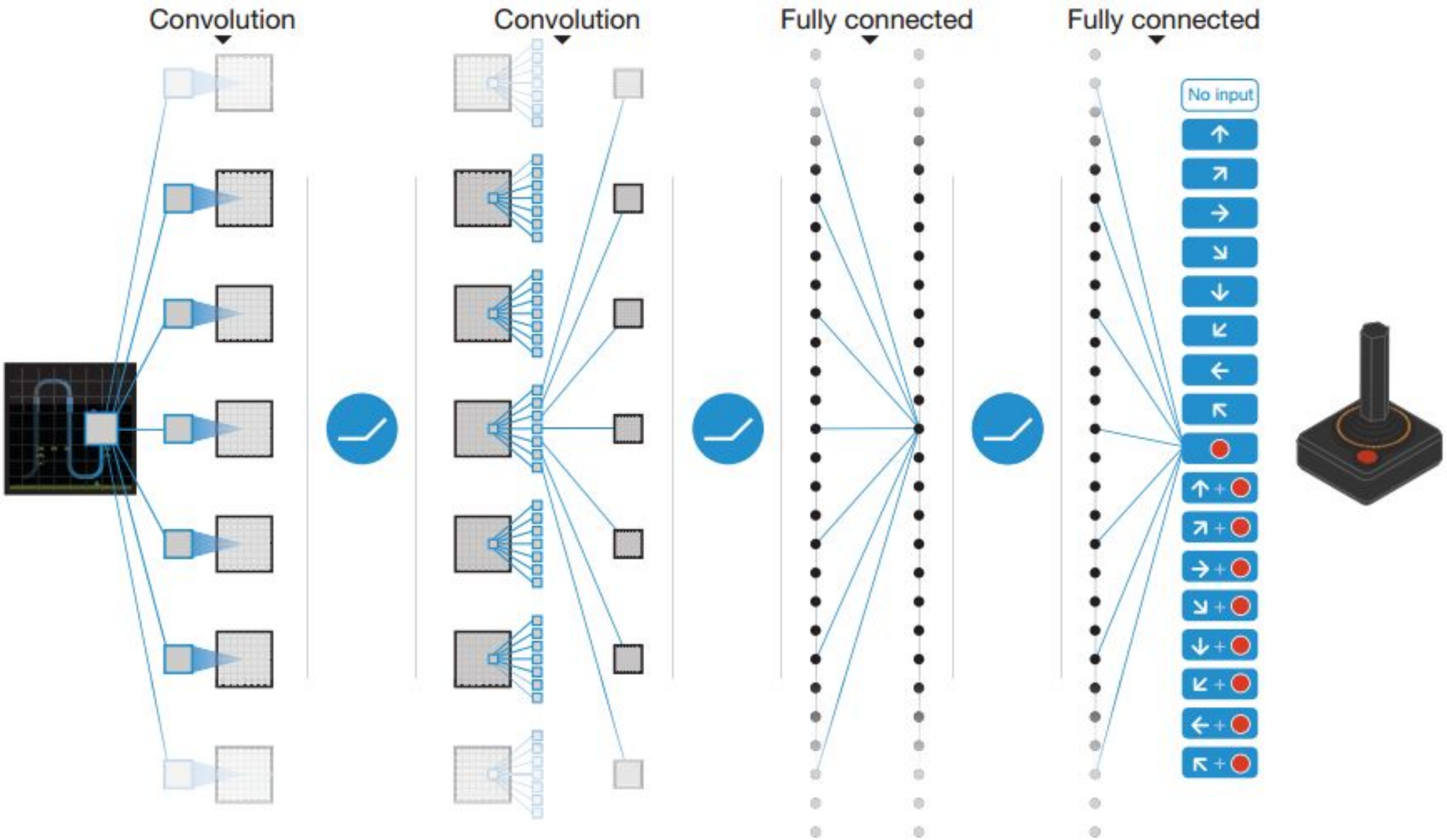
Deep Q-Learning Network (DQN)




Q. What about input features?

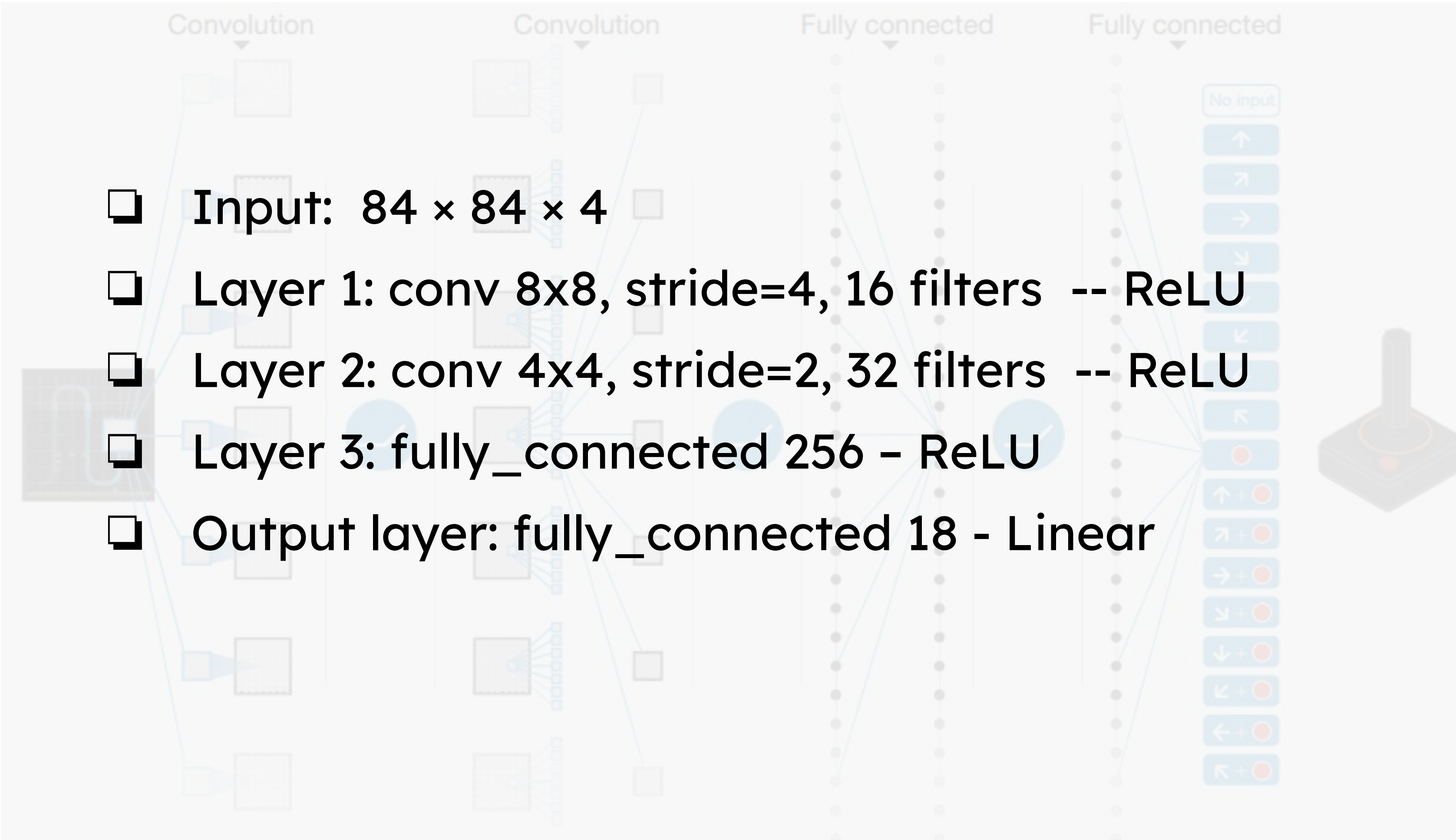
A. Learnt to play from video input from <u>scratch!</u>

DQN



Source: Deep Q Networks, Nature 2015

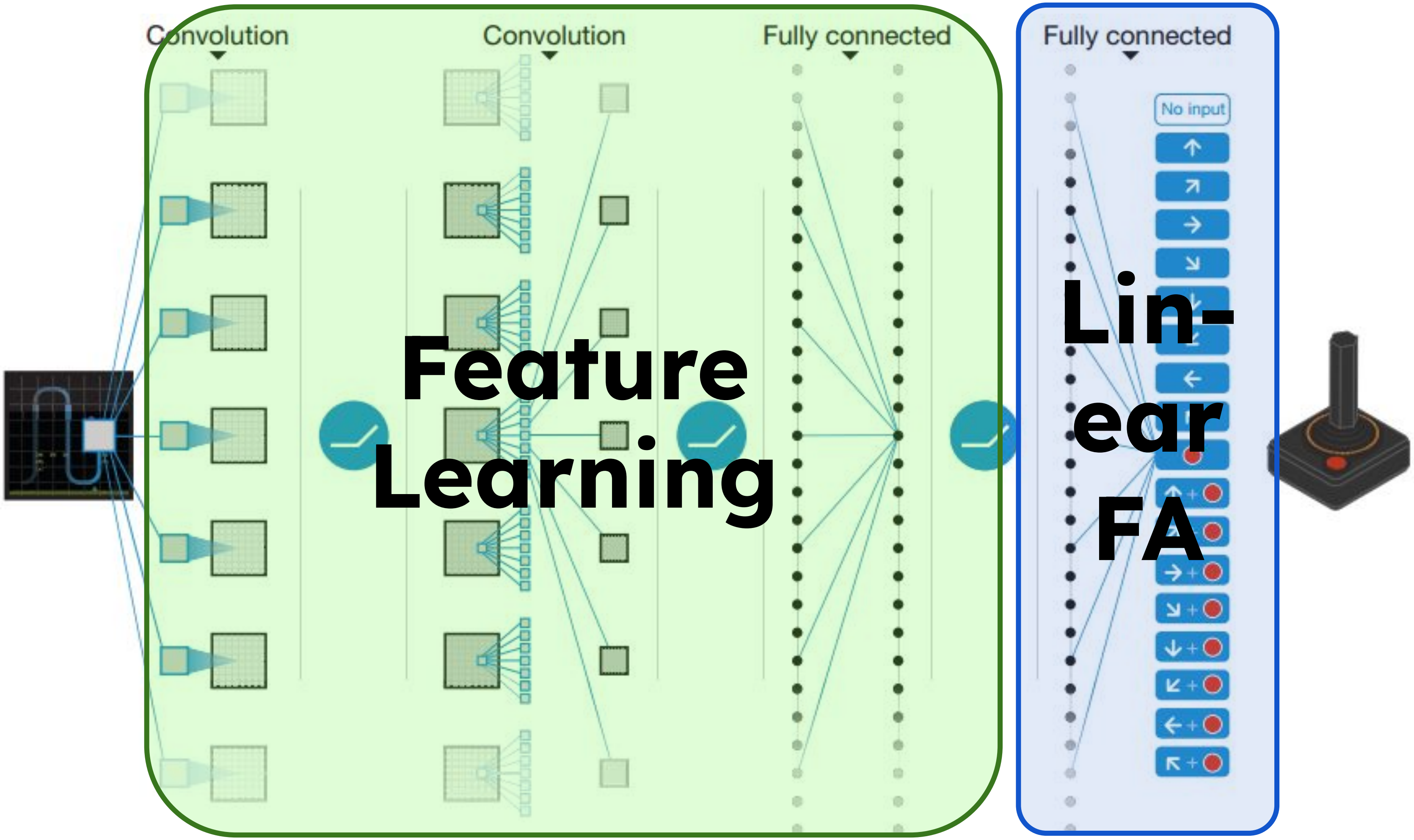
DQN



- Input: 84 × 84 × 4
- Layer 1: conv 8x8, stride=4, 16 filters -- ReLU
- Layer 2: conv 4x4, stride=2, 32 filters -- ReLU
- Layer 3: fully_connected 256 – ReLU
- Output layer: fully_connected 18 - Linear

Source: Deep Q Networks, Nature 2015

DQN

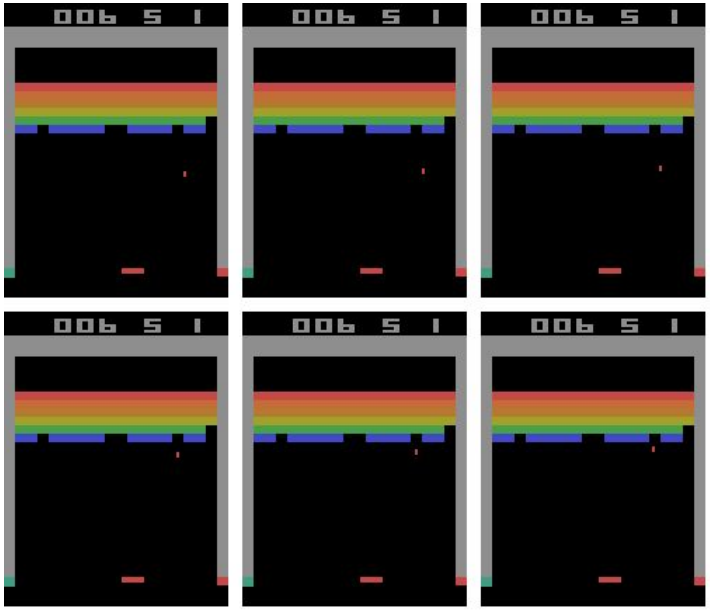


Source: Deep Q Networks, Nature 2015

Q-Network Learning

$$w_{t+1} = w_t - \frac{1}{2}\alpha\nabla_{w_t}\left[r_{t+1} + \gamma\max_a \hat{q}\left(s_{t+1}, a\right) - \hat{q}\left(s_t, a_t\right)\right]^2$$

- Divergence is an issue since the current network is used decide its own target
- Correlations between samples
- Non-stationarity of the targets



- How do we address these issues?
  - Replay Memory
  - Freeze target network

Q-Network Learning

- **Replay Memory/Buffer** : To remove correlations, we build data-set from the agent's experience

s_1, a_1, r_2, s_2
s_2, a_2, r_3, s_3
s_3, a_3, r_4, s_4
$\ldots$
$s_t, a_t, r_{t+1}, s_{t+1}$

$\rightarrow \quad s, a, r, s'$

$s_t, a_t, r_{t+1}, s_{t+1} \quad \rightarrow$

- **Frozen Target Network** : Sample experiences from dataset; w^- frozen (with periodic updates) to address non-stationarity

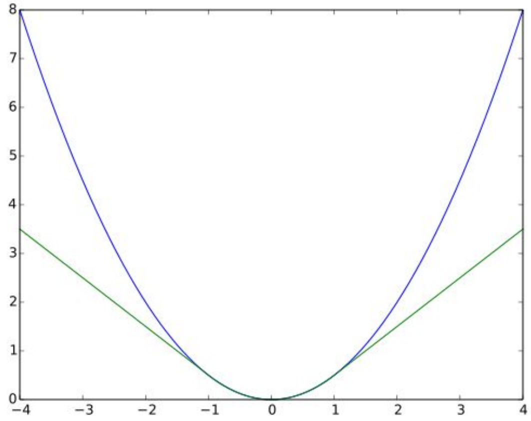
$$\left[\left\{r_{t+1} + \gamma \max_a \hat{q}\left(s_{t+1}, a; w^-\right)\right\} - \hat{q}\left(s_t, a_t; w\right)\right]^2$$

DQN

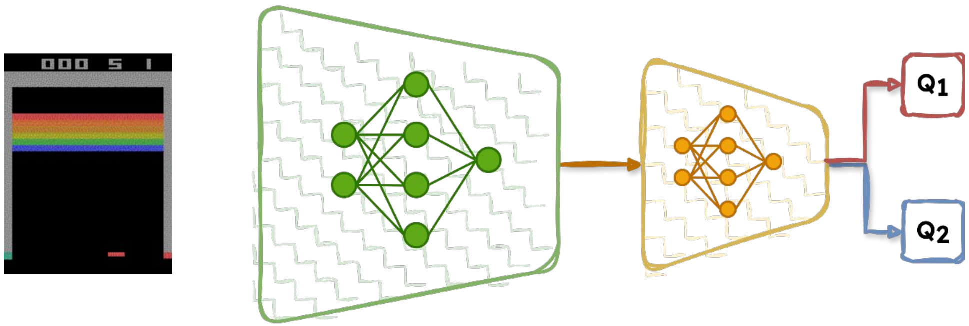
- Architecture:
  - Has a set of convolutional layers which act as feature extractors
  - These features are then passed through a series of fully connected layers
  - The output layer has |A| number of nodes which are used to calculate the Q-value for each action

- The network is updated using huber loss and not regular least squares loss

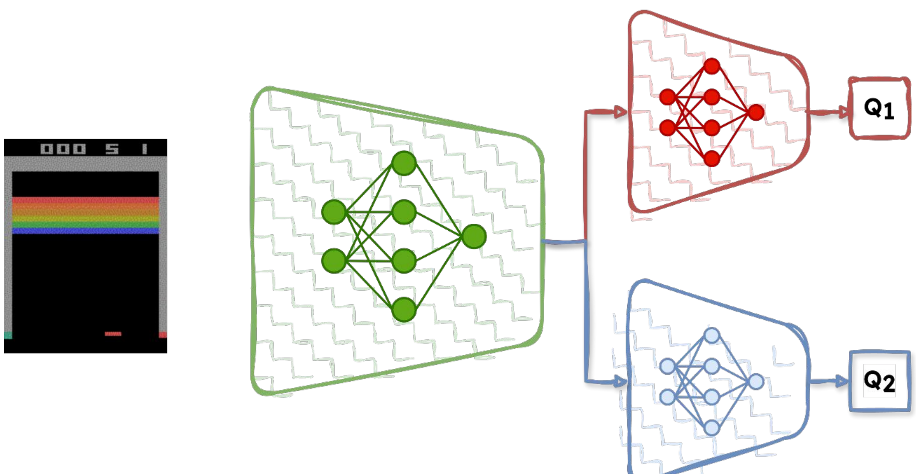
$$L_\delta(a) = \begin{cases} \frac{1}{2}a^2 & \text{for } |a| \le \delta, \\ \delta(|a| - \frac{1}{2}\delta), & \text{otherwise.} \end{cases}$$



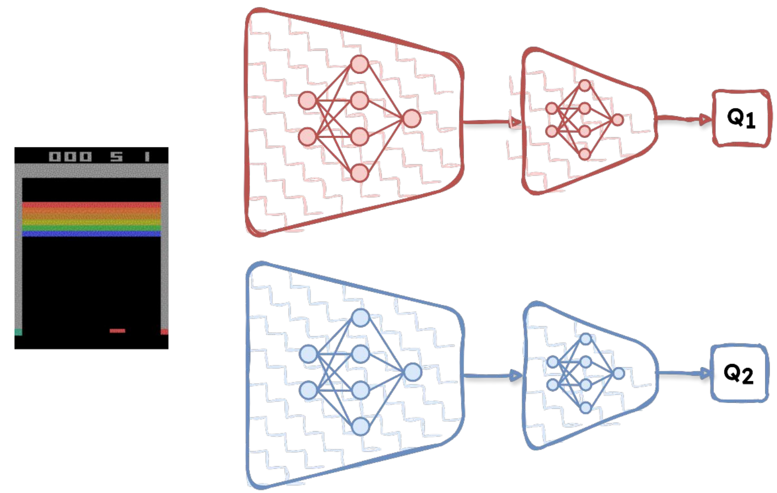
DQN - Design Choices



Type 1 : Fully Sharing



Type 2 : Semi-Sharing



Type 3 : No sharing