

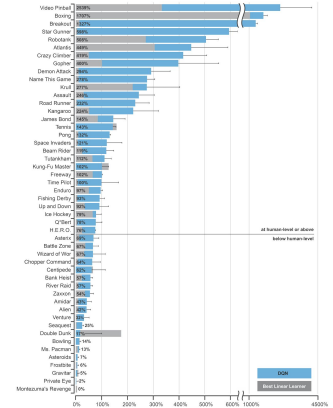
Lecture 7:  
Function Approximation,  
SGD, DQN  
B. Ravindran

# 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

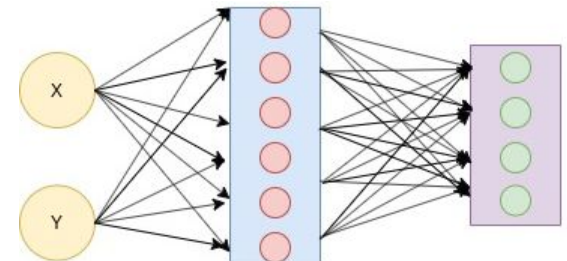
# Non-Linear Function Approximator

- Linear function approximators are very 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

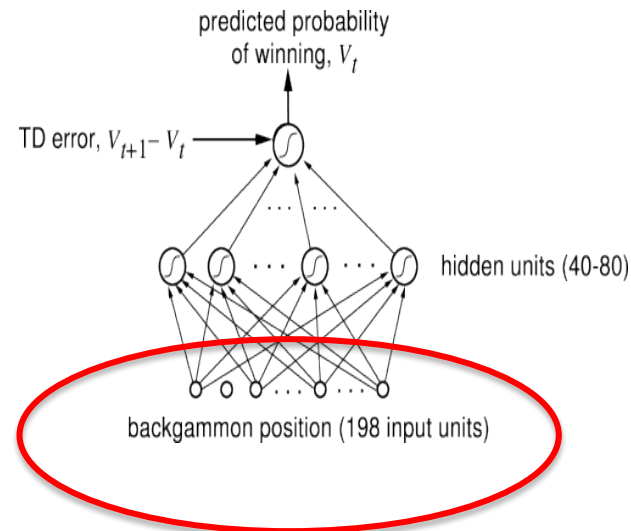
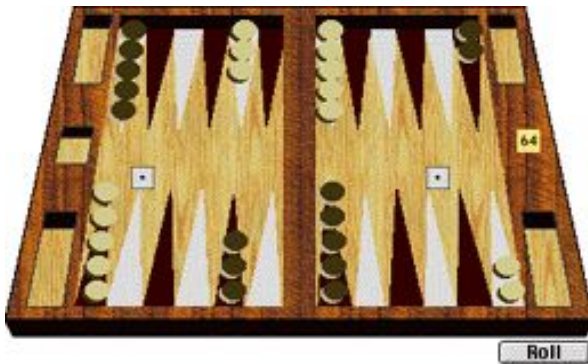
- **Gradient Descent** is a first-order iterative optimization algorithm for finding a local minimum of a differentiable function.
- In Gradient Descent, we compute the gradient of the function at the current point and take a step in the opposite direction. This is the direction of steepest descent.
- 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.

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

# Human Level Backgammon player

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.



# Semi Gradient Methods

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

$$\hat{q}(s_t, a_t) = \phi^T(s_t, a_t) \times w_t$$

$$\delta_t = r_{t+1} + \gamma \max_a \hat{q}(s_{t+1}, a) - \hat{q}(s_t, a_t) \quad \text{TD Error}$$

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

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

# Stochastic Gradient Descent

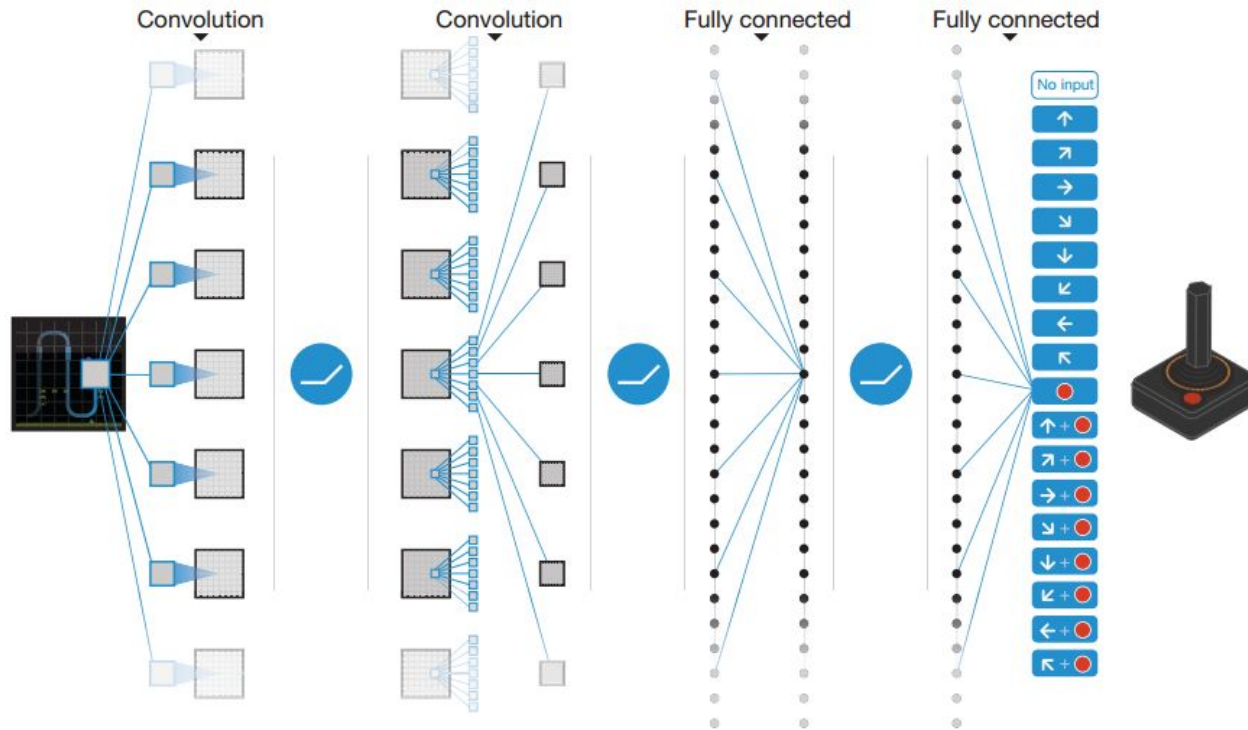
- In **Stochastic Gradient Descent**, the true gradient of the loss function is approximated by the gradient at a single example.
- In practice, we usually perform **Mini Batch Gradient Descent**, where we compute the gradient using a mini batch of examples. This allows for more efficient computation and smoother convergence.

# What about the features?

Learnt to play from video input from scratch!

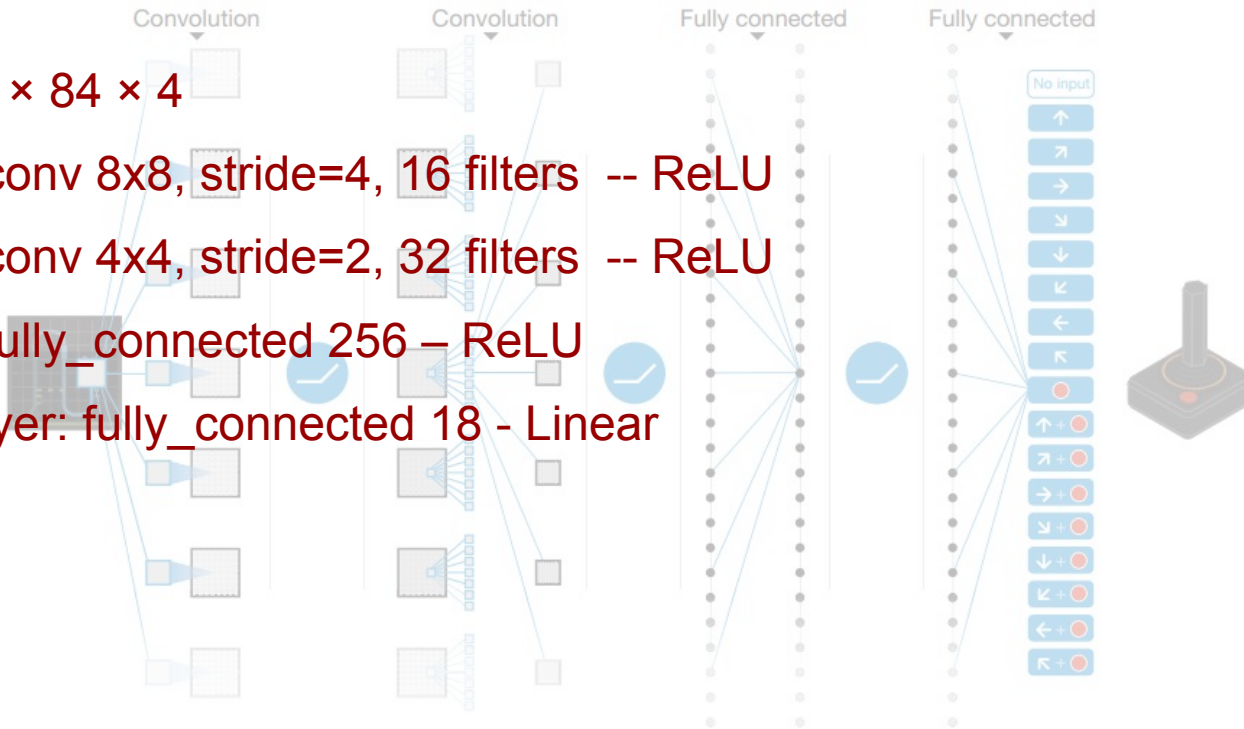


# Deep Q-Learning

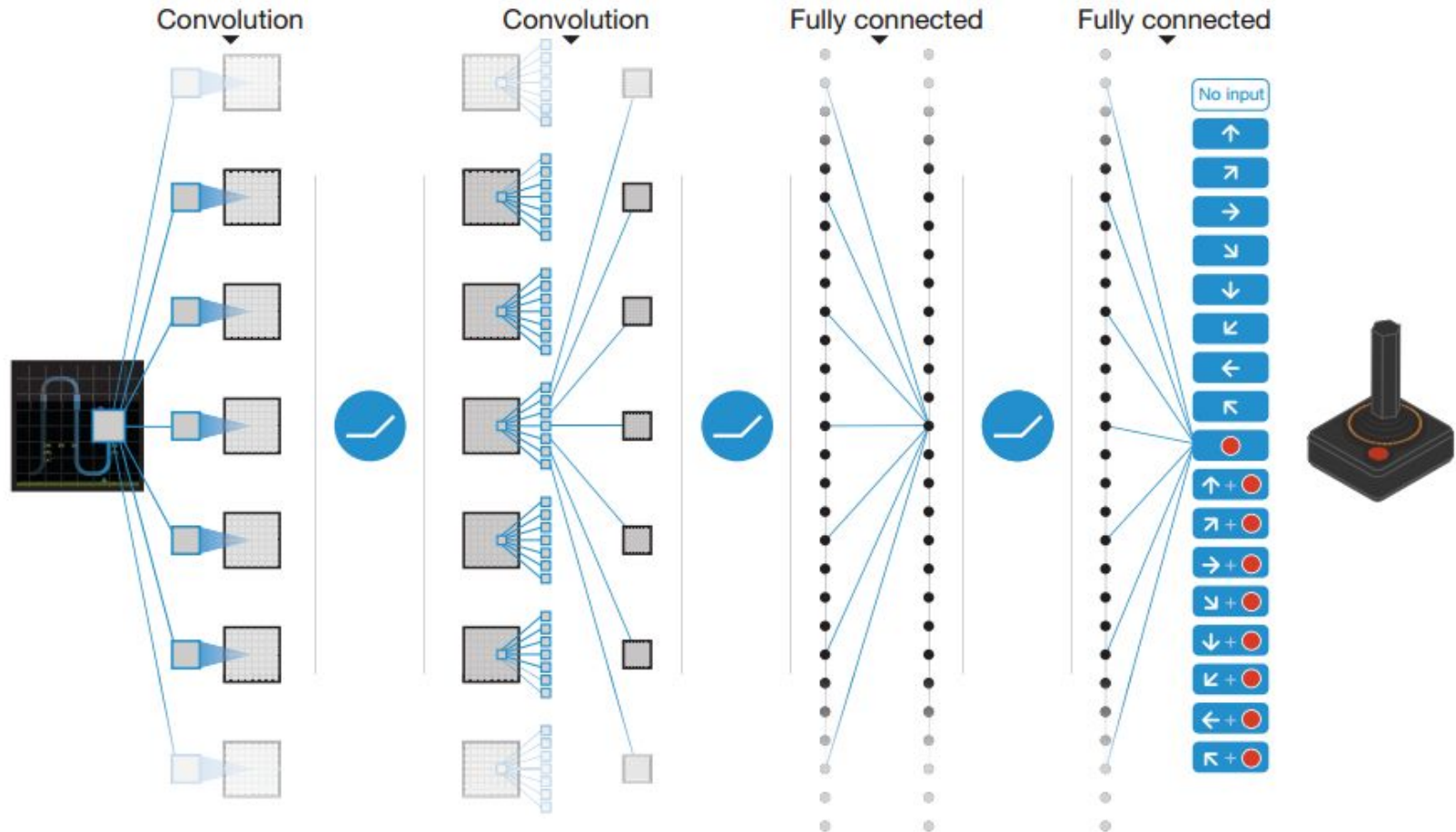


# Deep Q-Learning

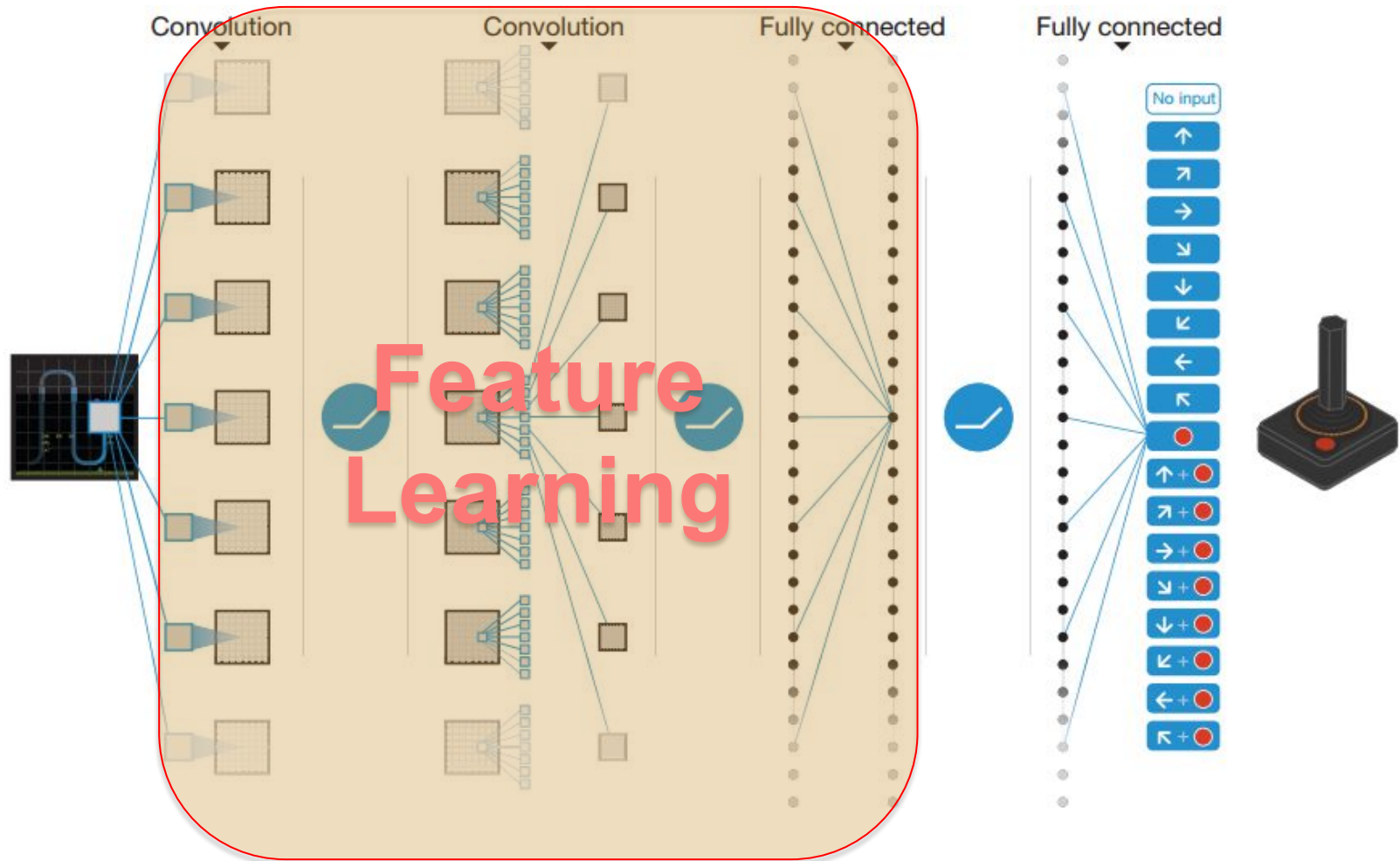
- Input:  $84 \times 84 \times 4$
- Layer 1: conv  $8 \times 8$ , stride=4, 16 filters -- ReLU
- Layer 2: conv  $4 \times 4$ , stride=2, 32 filters -- ReLU
- Layer 3: fully\_connected 256 -- ReLU
- Output layer: fully\_connected 18 - Linear



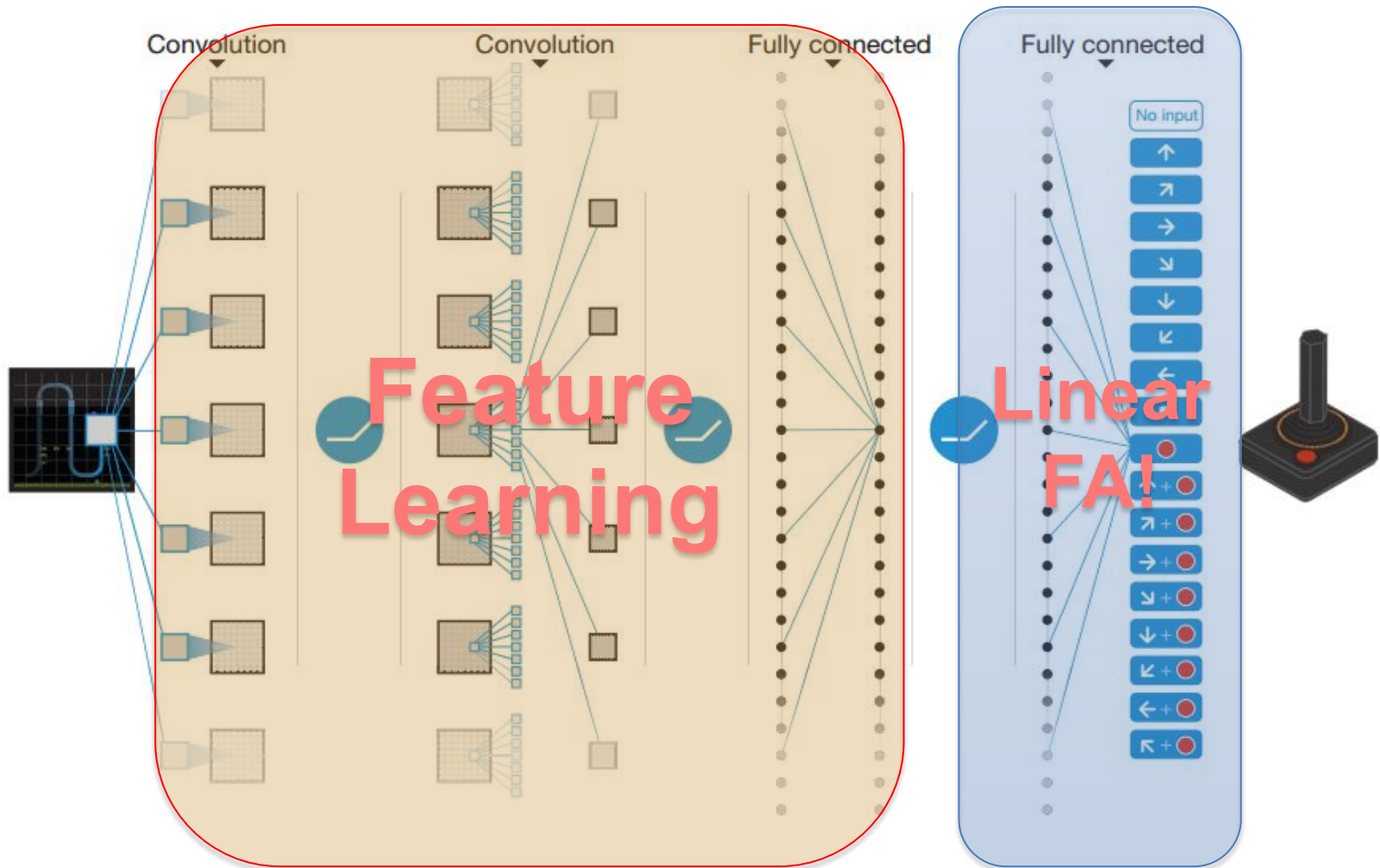
# Deep Q-Learning



# Deep Q-Learning



# Deep Q-Learning



# Q-Network Learning

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

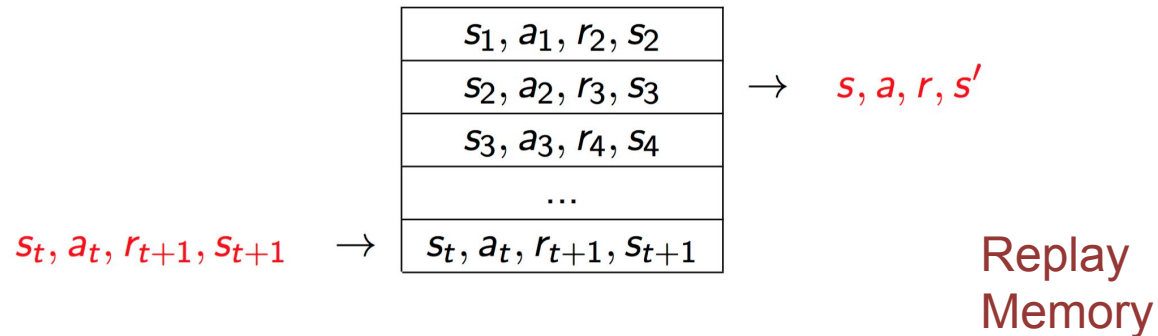
**Divergence** is an issue since the current network is used to decide its own target

- Correlations between samples.
- Non-stationarity of the targets.
- How do we address these issues?

**Replay Memory & Freezing target network**

# Q-Network Learning

To remove correlations, we build data-set from the agent's experience



Sample experiences from dataset;  $w^-$  frozen (with periodic updates) to address non-stationarity

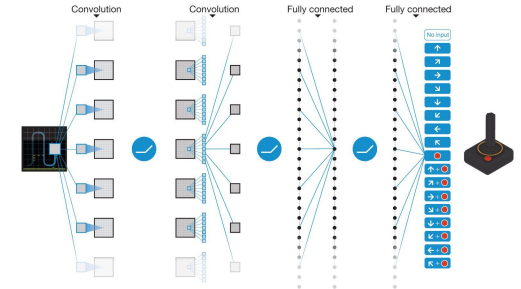
$$\left[ \left\{ r_{t+1} + \gamma \max_a \hat{q}(s_{t+1}, a; w^-) \right\} - \hat{q}(s_t, a_t; w) \right]^2$$

Freeze Target Network

# Deep-Q Networks

## 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 above network is updated using huber loss and not regular least squares loss.

Below is the regular expression for huber loss.

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

