Faculté des **sciences économiques** et de **gestion**
Université de Strasbourg

Reinforcement Learning

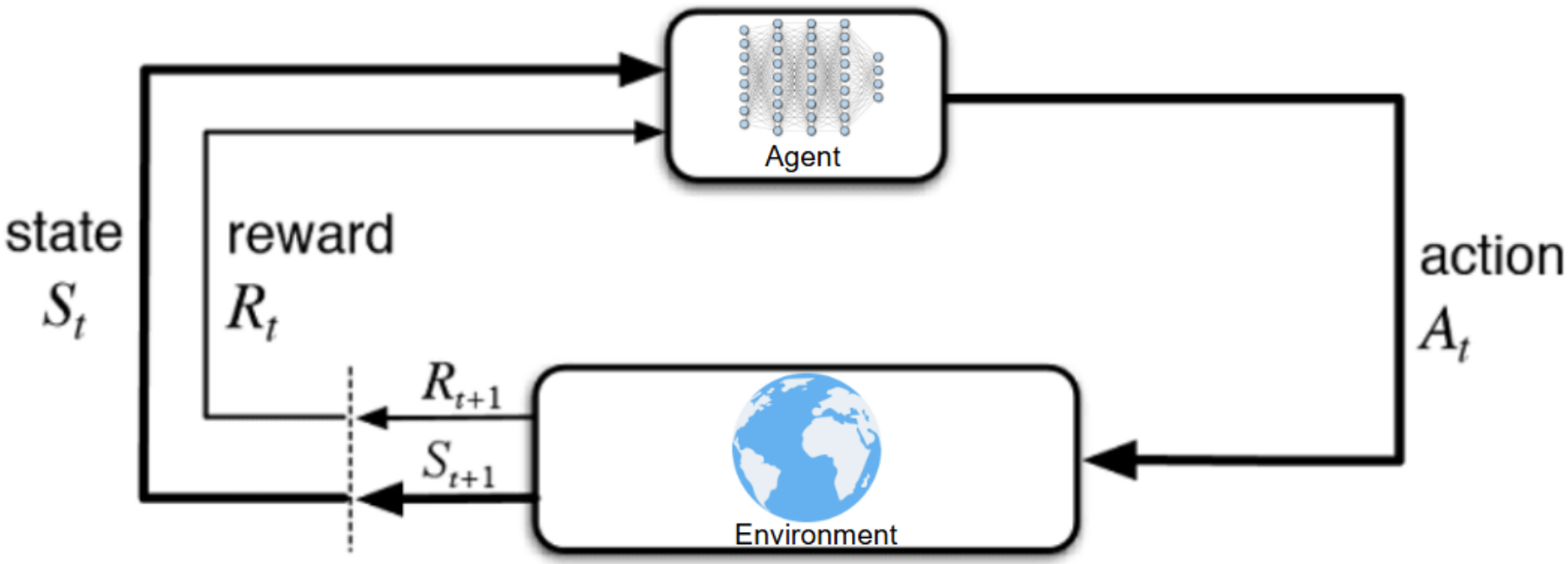
UE 2 Machine learning

Nattirat Mayer

Fall-Winter Semester 2025-26

Chapter 7. Deep Reinforcement Learning

7.1 Intro to Deep Reinforcement Learning



Recall, Q-Learning estimates $Q(s, a)$ as

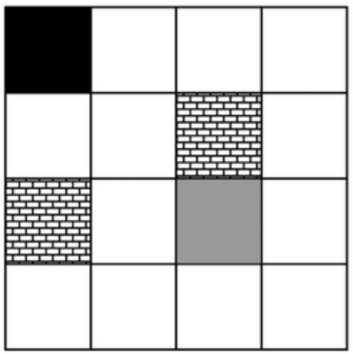
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \Big]$$

When selecting an action, the agent uses

$$A_t = \arg\max_a Q(S_t, a)$$

Recall, TD (non neural network) is a tabular learning with Q-table with num of states x num of actions.

a)



Maze

b)

	$a1$	$a2$	$a3$	$a4$
$s1$	0	0	0	0
$s2$	0	0	0	0
$s3$	0	0	0	0
$s4$	0	0	0	0

Q-table (Initialization)

c)

	$a1$	$a2$	$a3$	$a4$
$s1$	−1	−1	2	0
$s2$	3	−6	5	1
$s3$	2	9	−3	5
$s4$	0	5	3	8

Q-table (Training)

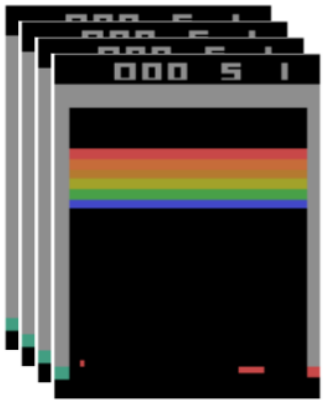
Explorer Trap Treasure

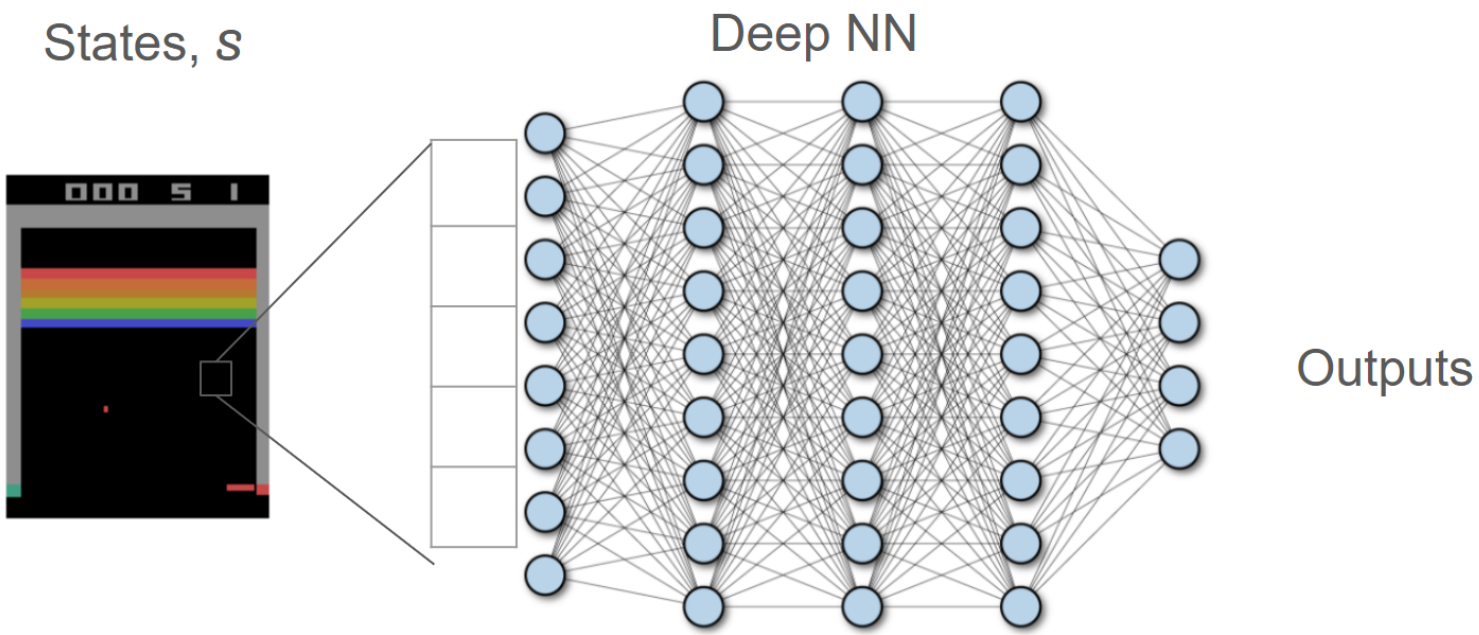
In practice, iterating $Q(s, a)$ is impractical because if the state space becomes very large. For example,

- state = screen pixel
  - Image size 84×84
  - Consecutive 4 images
  - Grayscale with 256 gray levels

Then the state space becomes:

$$256^{84 \times 84 \times 4} = \text{unimaginably huge!}$$



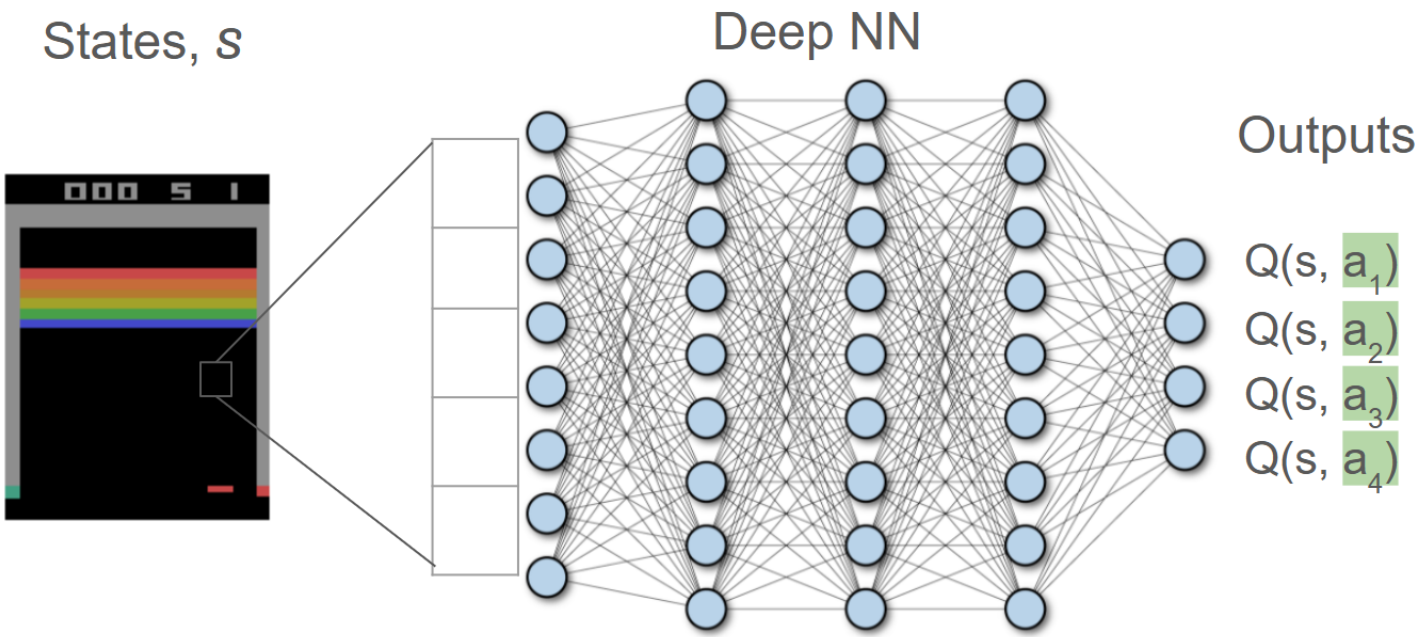

States, s
Deep NN
Outputs

How do our agents learn?

In this course, we will look at

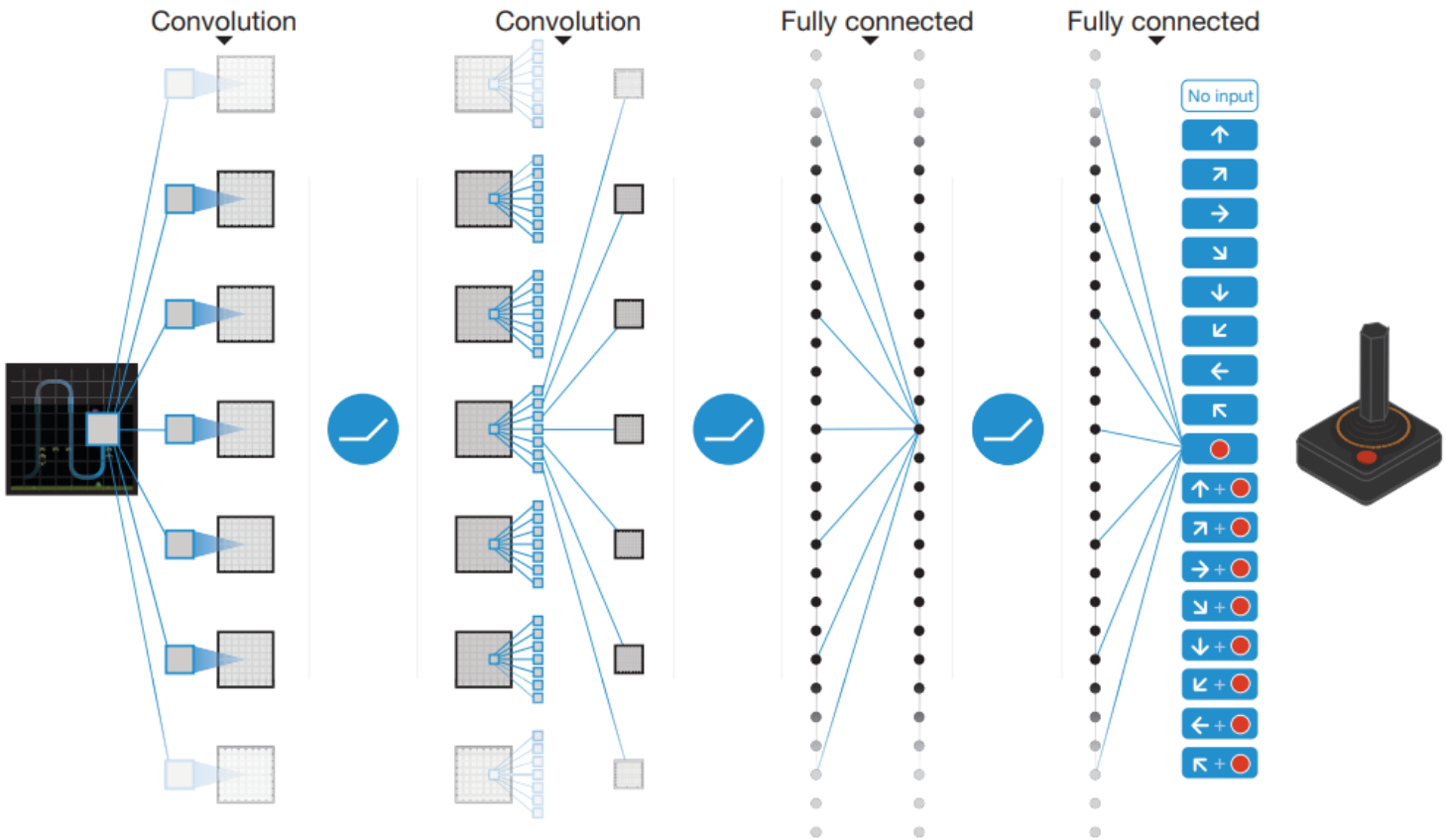
- Deep Q-Learning (DQN)
- Policy Gradient (PG)

7.2 Deep Q-Learning (DQN)



Simplified Deep Q-Learning (DQN)

Deep Q-Learning (DQN) - Mnih et al. (2015)

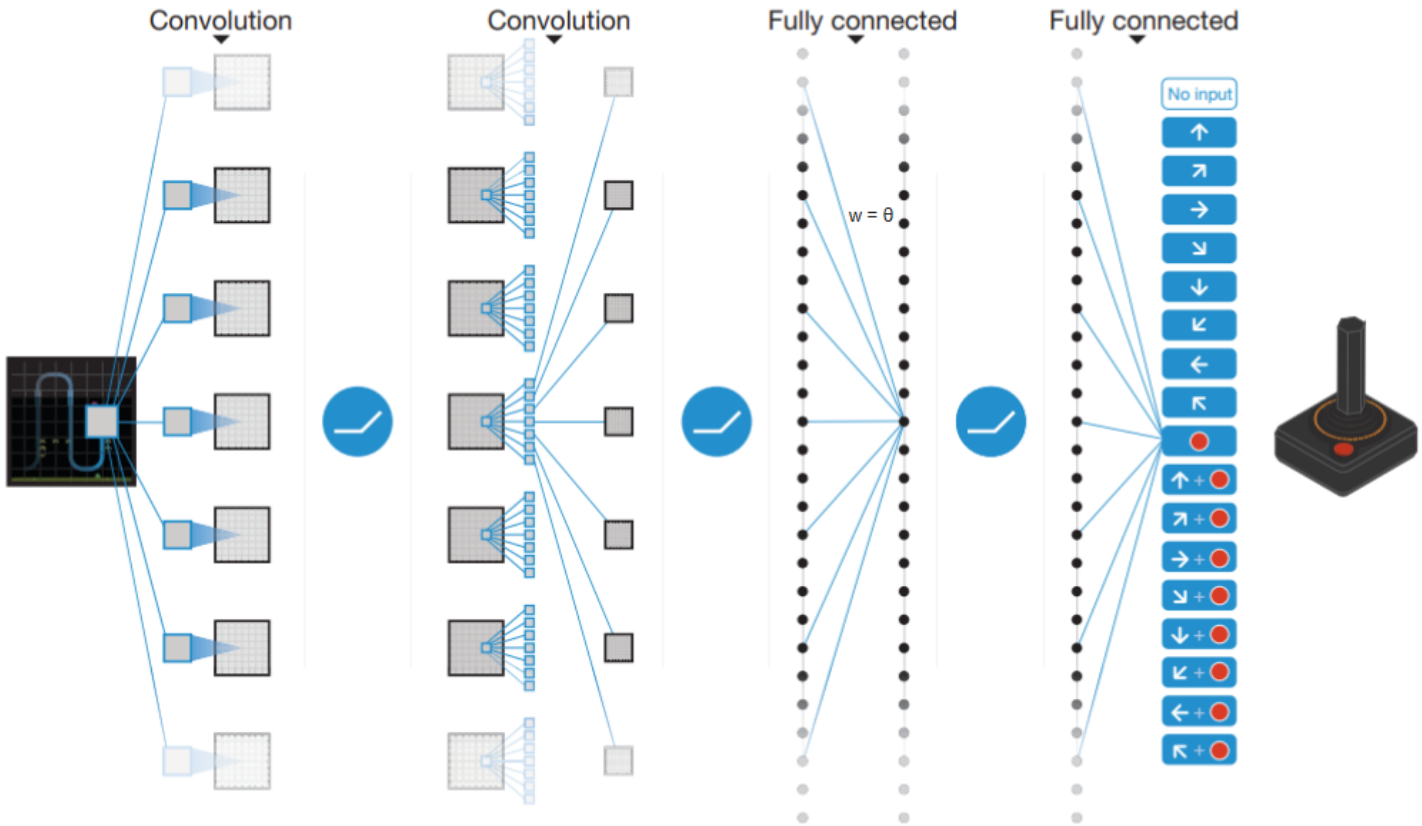


Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves et al. "Human-level control through deep reinforcement learning." nature 518, no. 7540 (2015): 529-533.

Deep Q-Learning (DQN) extends Q-Learning by approximating the action-value function with a deep neural network. States space $\mathcal{S}$ could be massive (e.g. Chess ($\mathcal{S} \approx 10^{80}$)). Thus, we introduce a parameter θ which has much lower dimensionality than $\mathcal{S}$

$$Q(s, a) \approx Q(s, a; \theta)$$

In DQN, we parameterize an approximate value function $Q(s, a; \theta)$ using the deep convolutional neural network, in which θ are the parameters (that is, weights w) (Mnih et al., 2015)



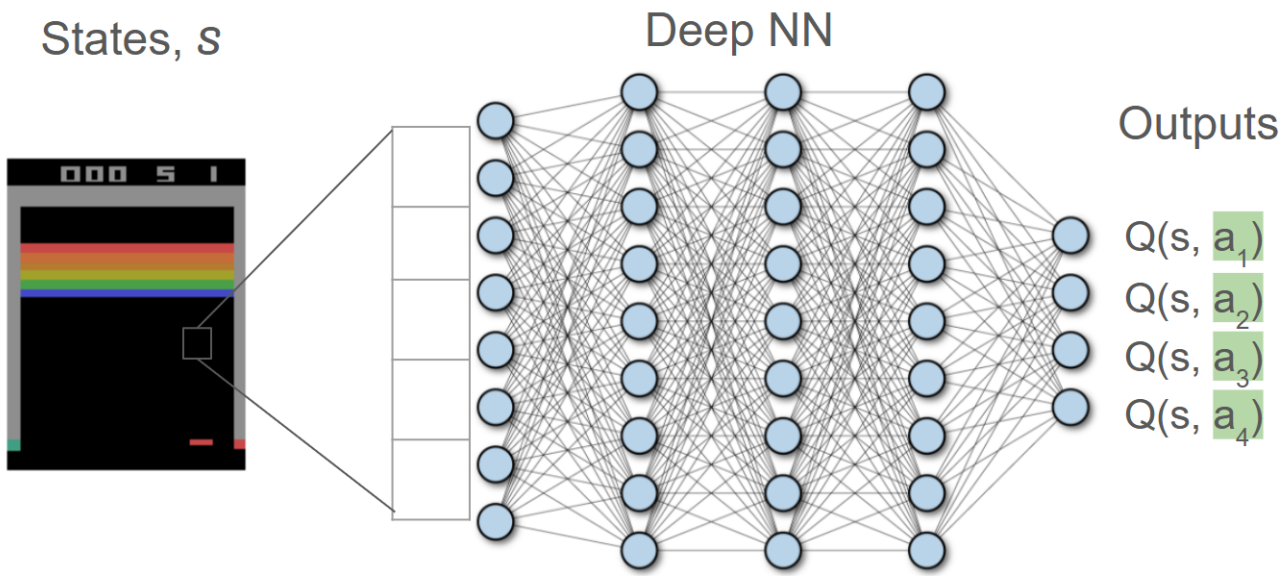
Loss Function of DQN

The Deep Q-learning update minimizes the following loss function:

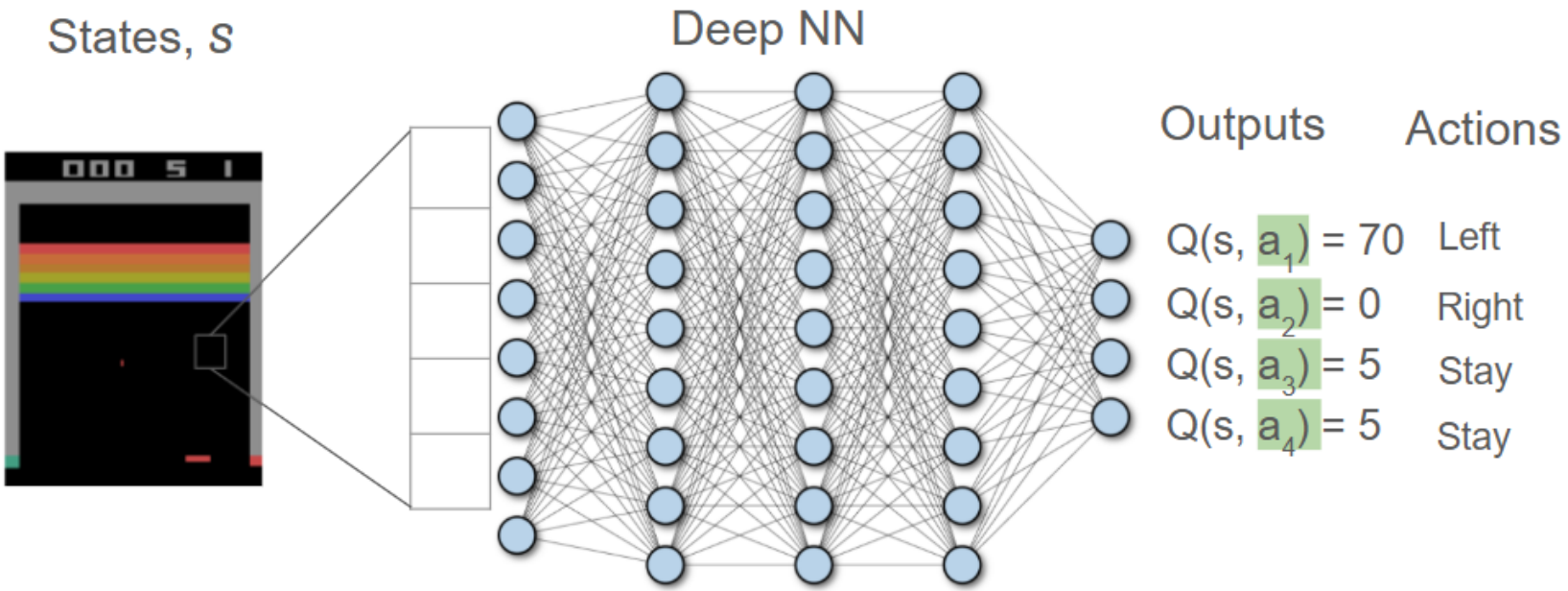
$$L(\theta) = \mathbb{E}\left[\left(R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \theta) - Q(S_t, A_t; \theta)\right)^2\right]$$

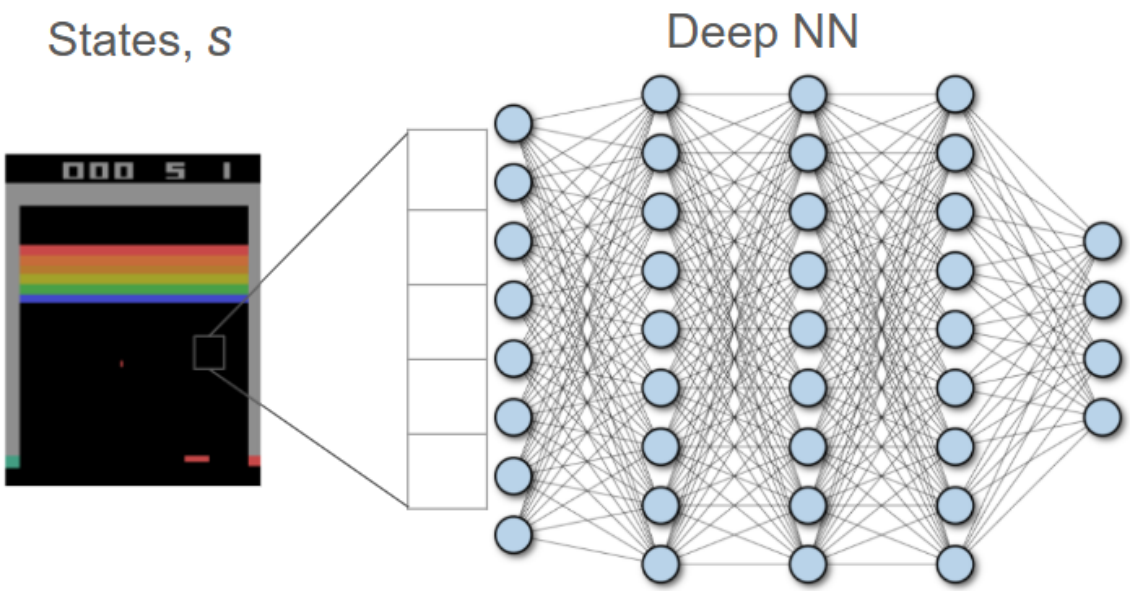
Mnih et al. (2015) trains large neural networks using a reinforcement learning signal and stochastic gradient descent to optimize θ that give optimal $Q(s, a)$ function that minimize the loss function $L(\theta)$

Recall that in **DQN**, the agent learns (approximates) the $Q(s, a)$,

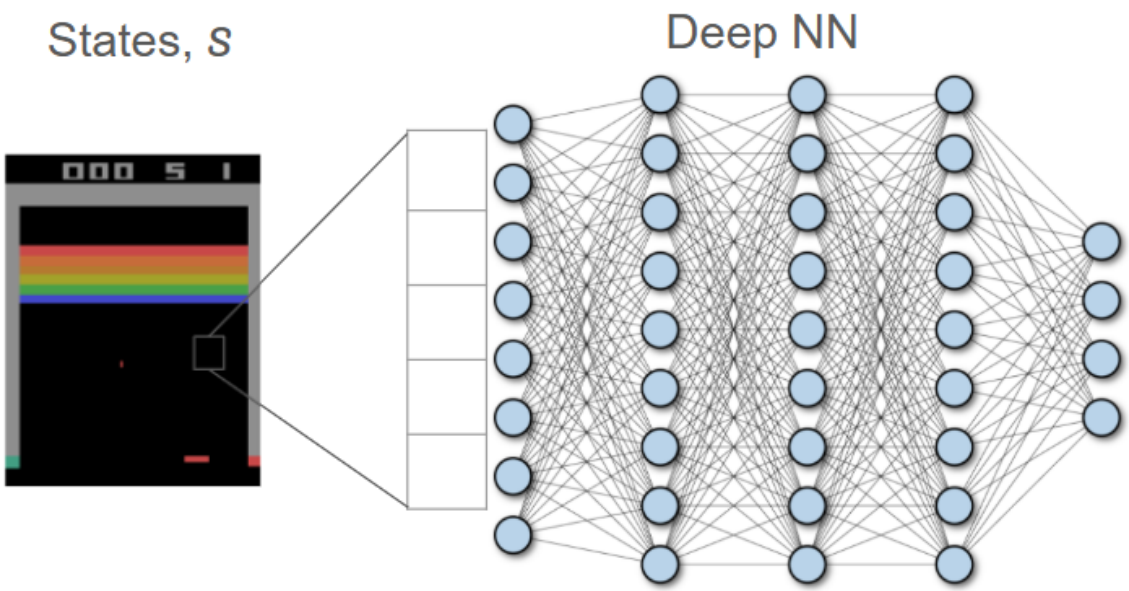


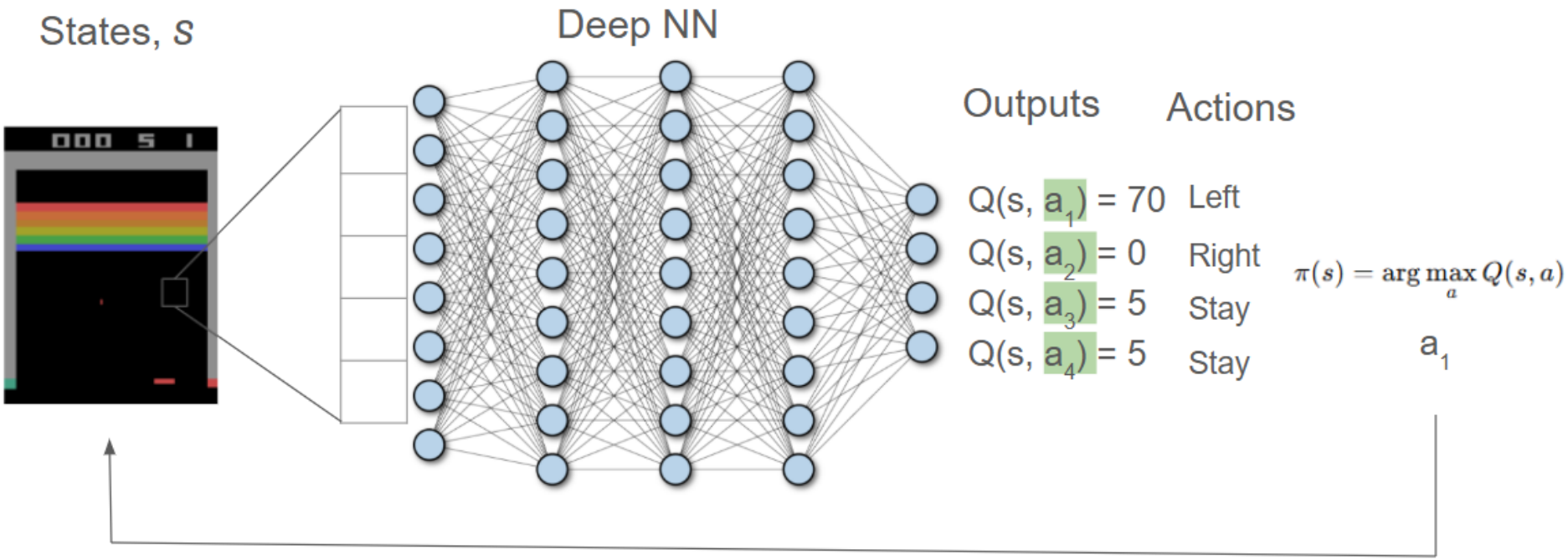
Simplified DQN


States, s
Deep NN
Outputs
Actions
$Q(s, a_1) = 70$
Left
$Q(s, a_2) = 0$
Right
$Q(s, a_3) = 5$
Stay
$Q(s, a_4) = 5$
Stay



$$\pi(s) = \arg\max_a Q(s, a)$$

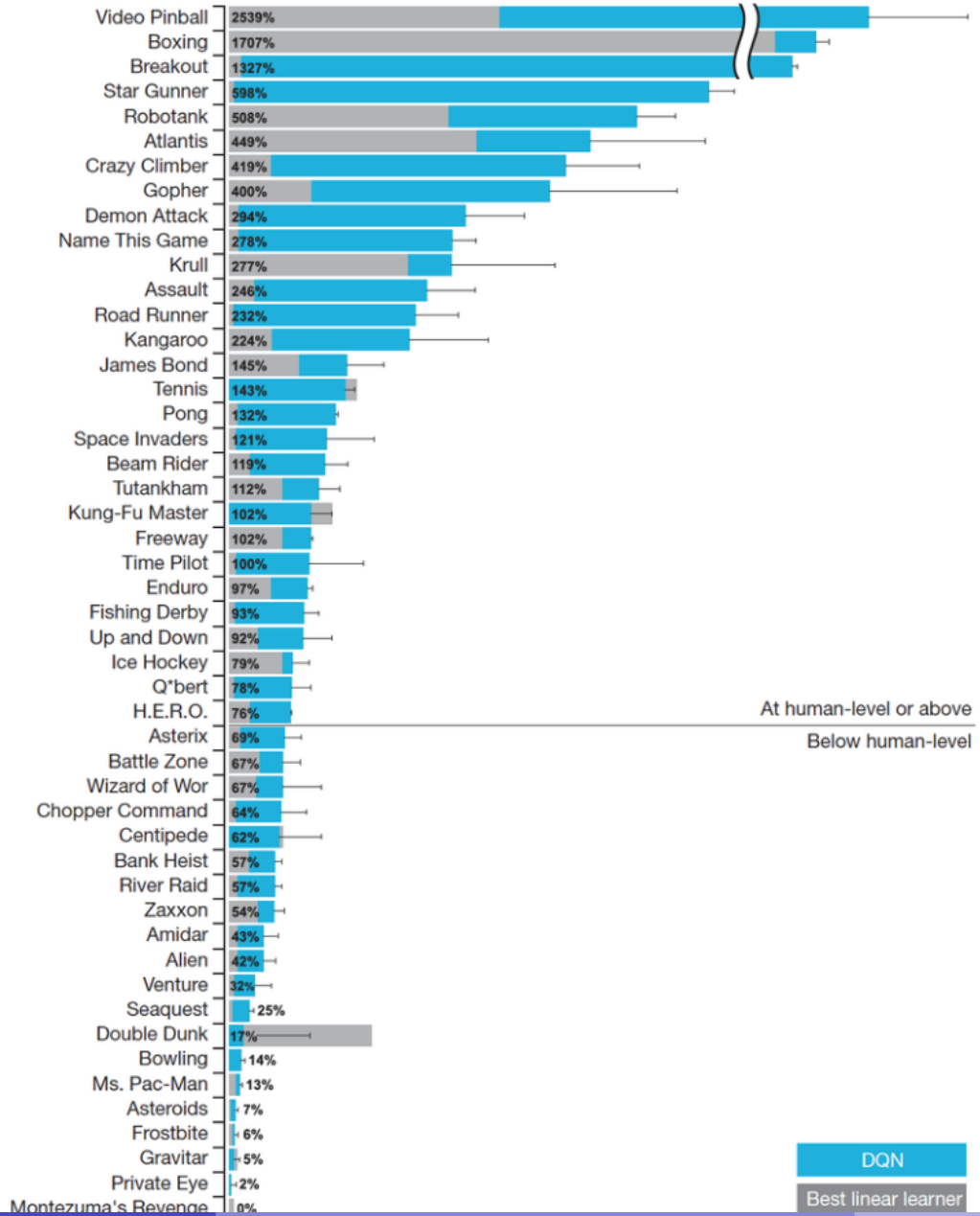

States, s
Deep NN
Outputs
Actions
Q(s, a1) = 70
Left
Q(s, a2) = 0
Right
Q(s, a3) = 5
Stay
Q(s, a4) = 5
Stay
$\pi(s) = \arg\max_a Q(s, a)$
a1



The agent sends the selected action back to the environment and receives the next state and reward. θ are also updated.

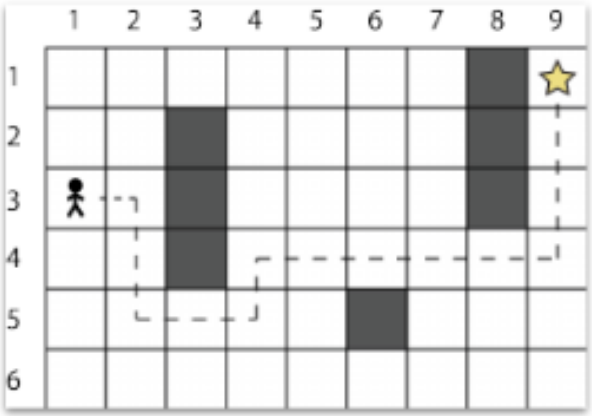
Question: Where can we implement Exploration-Exploitation Trade-Off in DQN? (e.g. ϵ-greedy)

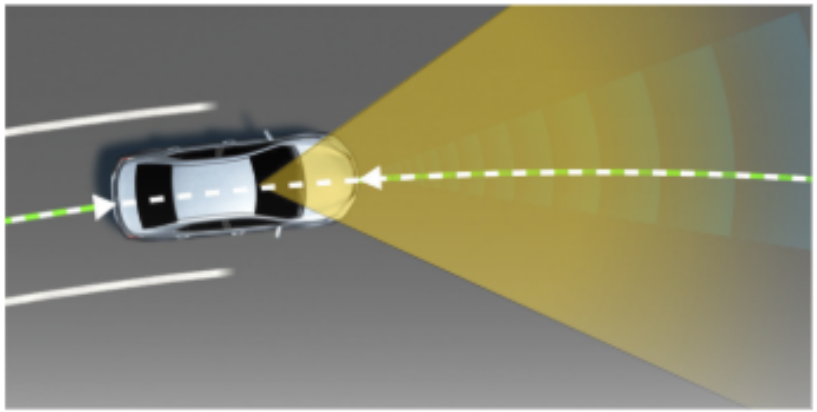
Success of DQN



Downsides of Deep Q-Learning

- DQN cannot handle continuous action spaces (e.g. in investment portfolio, in self-driving cars etc.)





Downsides of Deep Q-Learning (continue)

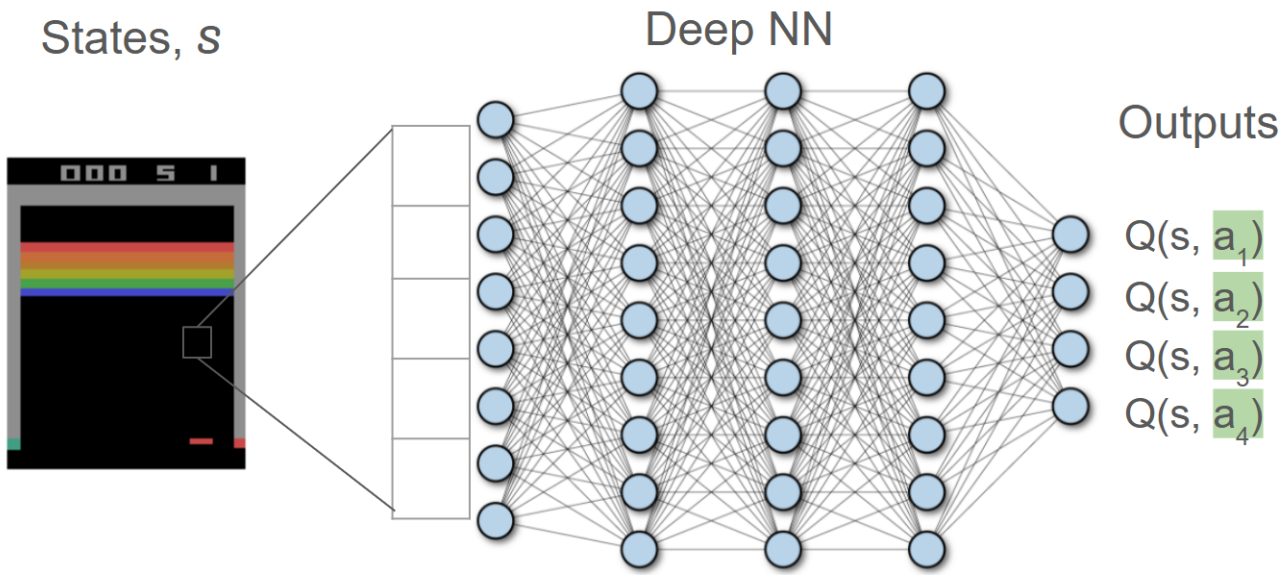
- DQN cannot learn stochastic policies since the policy is deterministically computed from the Q function $Q(s, a)$ because we choose the action from

$$\pi(s) = \arg\max_a Q(S_t, a)$$

Thus, given a state s, DQN will always choose the same action, resulting in a deterministic policy.

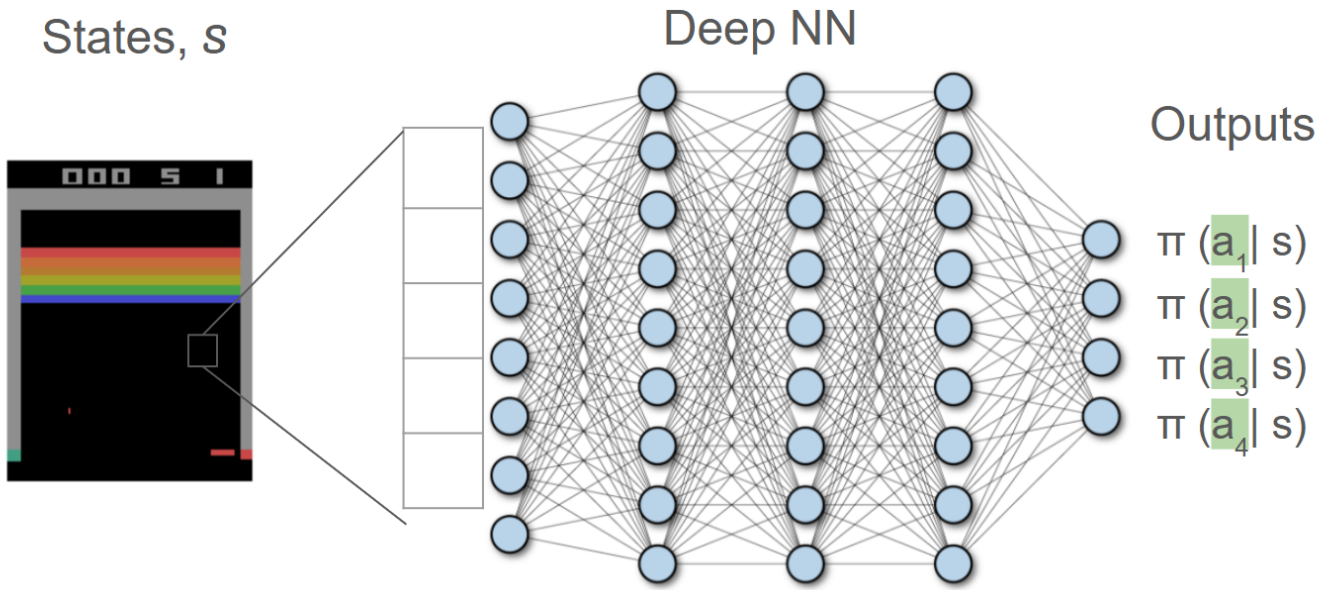
So, what could we do about this?

Recall that in **DQN**, the agent learns the $Q(s, a)$,



7.3 Policy Gradient (PG)

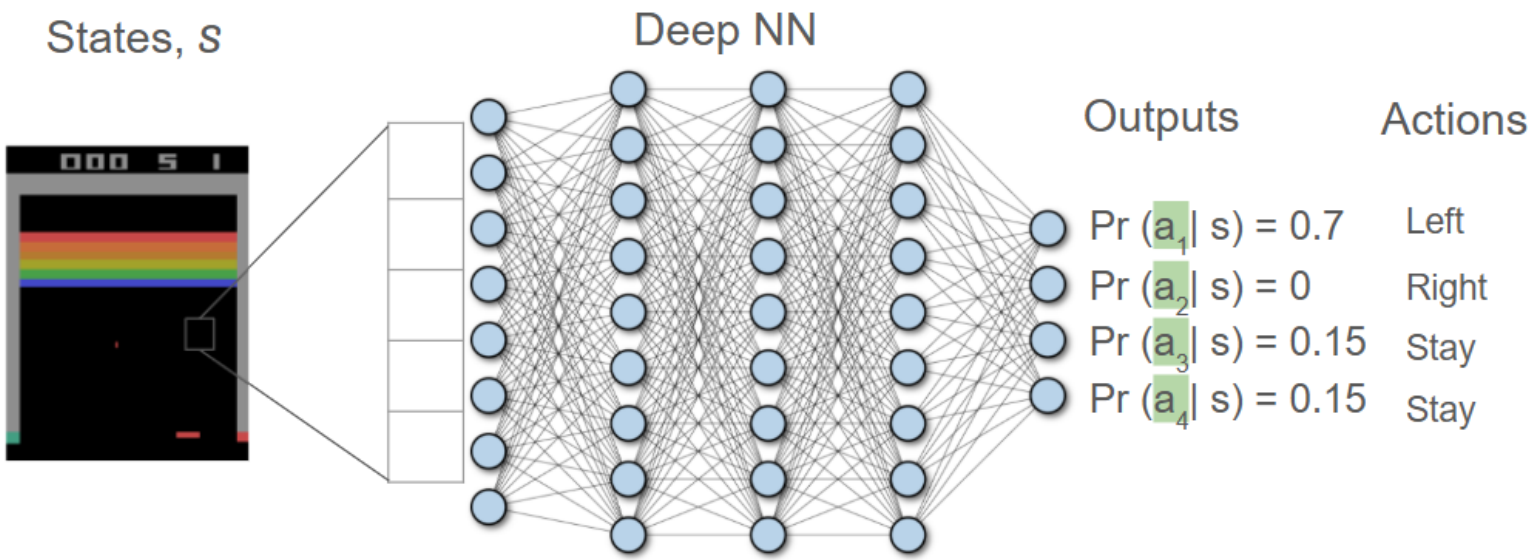
In **Policy Gradient** methods, we instead learn the **policy directly**

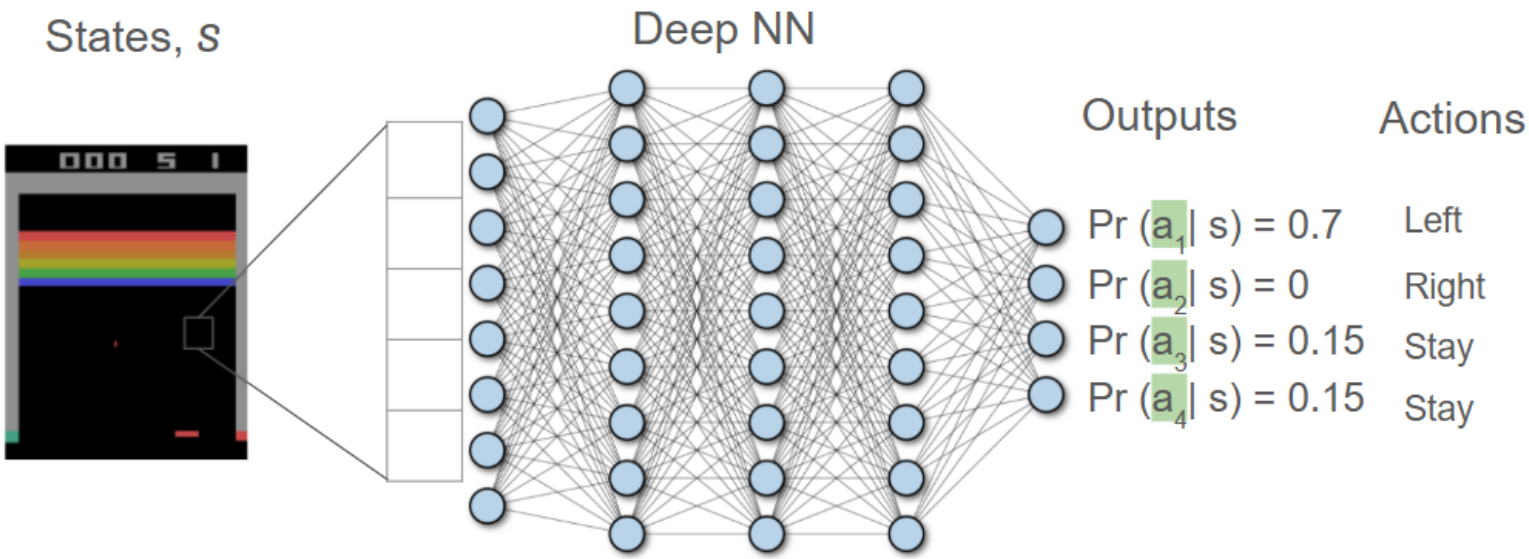


Simplified Policy Gradient (PG)

$$\pi_\theta(a|s)$$

where θ are the parameters of a neural network that outputs a probability distribution over actions.


States, s
Deep NN
Outputs
Actions
Pr (a_1| s) = 0.7
Left
Pr (a_2| s) = 0
Right
Pr (a_3| s) = 0.15
Stay
Pr (a_4| s) = 0.15
Stay



Now with these probability $\pi(a|s)$, we could

- Sample from that probability distributions which mean we could use concurrently with ϵ-greedy..Exploration and Exploitation
- **Account for continuous action spaces**

- The goal is to find θ that maximizes the expected return. Equivalently, we minimize the loss:

$$L(\theta) = -\log \pi_\theta(a|s) \cdot G_t$$

```
chosen_probs = tf.gather_nd(probs, tf.stack([indices, acts], axis=1))
loss = -tf.reduce_mean(tf.math.log(chosen_probs + 1e-8) * returns) #
```

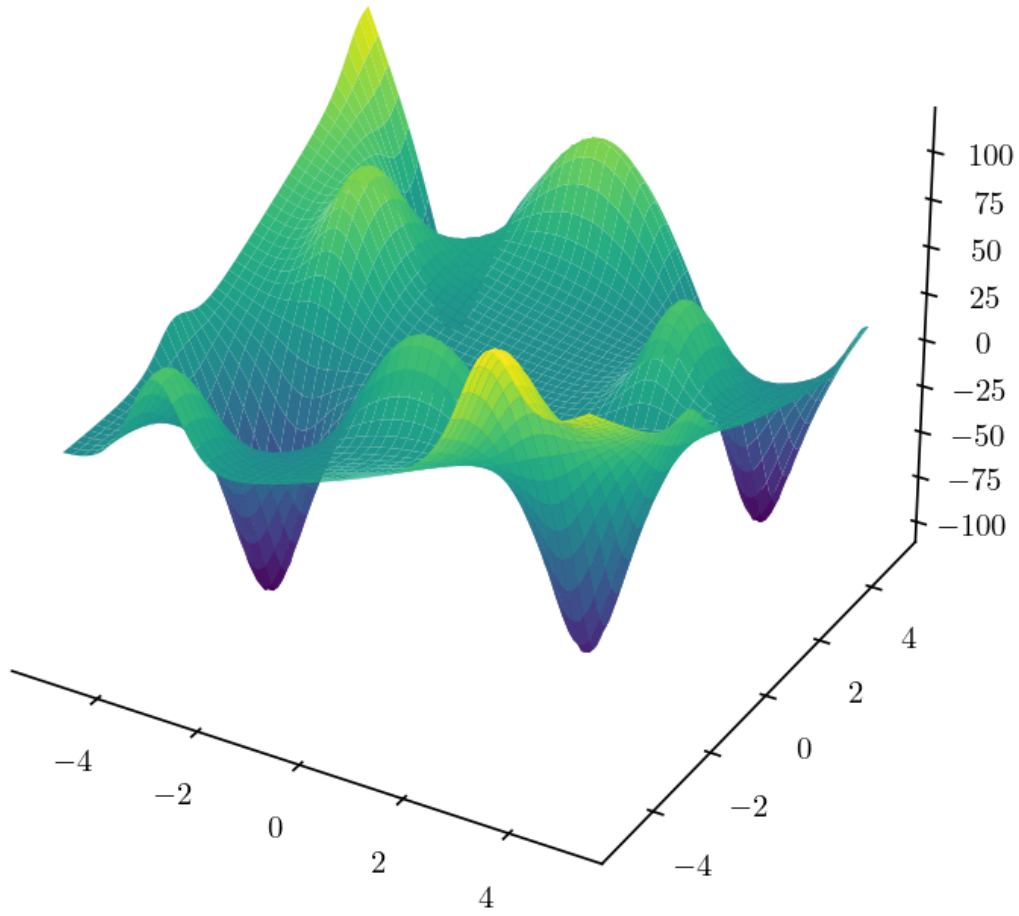
- We use **gradient descent** to update the policy parameters:

$$\begin{aligned}\theta' &= \theta - L(\theta) \\ &= \theta + \alpha \nabla_\theta \log \pi_\theta(a|s)\, G_t\end{aligned}$$

```
grads = tape.gradient(loss, self.model.trainable_variables)
self.optimizer.apply_gradients(zip(grads, self.model.trainable_variables)) # performs gradient descent
```

- The gradients ∇ are computed using **backpropagation** through the policy network.
- For the policy gradient algorithm, we can implement this using the Python package `REINFORCE`.

Loss Optimization - Gradient Descent




```
with tf.GradientTape() as tape:
    q = self.model(s)
    q_a = tf.reduce_sum(q * tf.one_hot(a, self.n_actions), axis=1)
    loss = tf.reduce_mean(tf.square(targets - q_a))
grads = tape.gradient(loss, self.model.trainable_variables)
self.optimizer.apply_gradients(zip(grads, self.model.trainable_variables))
```

Gradient Descent Steps

1. **Initialize parameters** θ_1, θ_2
2. **Compute gradient:**

$$\nabla_\theta L(\theta) = \frac{\partial L}{\partial \theta}$$

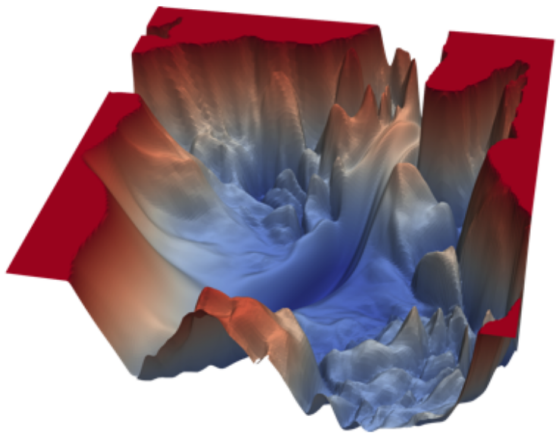
   This gives the direction of **steepest increase** in the loss.
3. **Update parameters (step opposite gradient):**

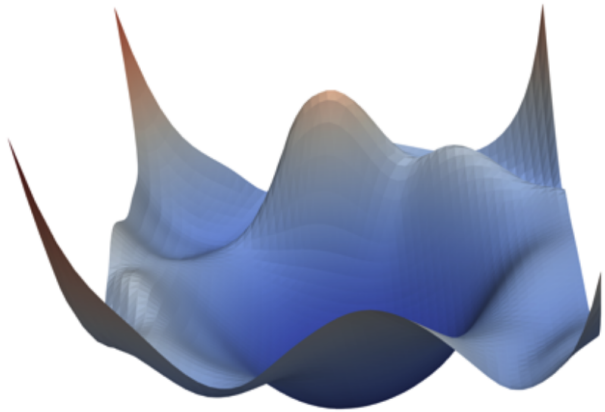
$$\theta \leftarrow \theta - \alpha \nabla_\theta L(\theta)$$

   where α is the learning rate.
4. **Repeat** over many iterations (batches/episodes) until the loss converges or performance stabilizes.

Downsides of Policy Gradient (PG) Methods

- **Sensitive to Learning Rate:** Too large α may cause divergence (jump around and fail to learn); too small α slows convergence.
- **Local Optima:** Updates are local; may converge to suboptimal policies in complex





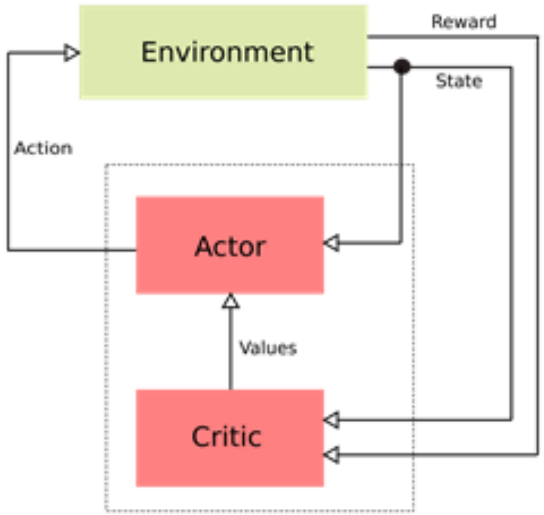
DQN Vs PG

- **Complex world:** If $Q(s, a)$ is too complex to be learned, DQN fails but PG will still learn a good policy
- **Speed:** PG is faster in convergence than DQN
- **Stochastic Policies:** PG is able to estimate stochastic policies. DQN cannot
- **Data:** PG needs more data. PG = sample inefficient

7.4 Actor-Critic Methods

Concept:

- Combines DQN and PG approaches.
- Two neural networks (Actor and Critic):
  - Actor = policy-based (sample actions from policy)
  - Critic = measures how good the chosen action is

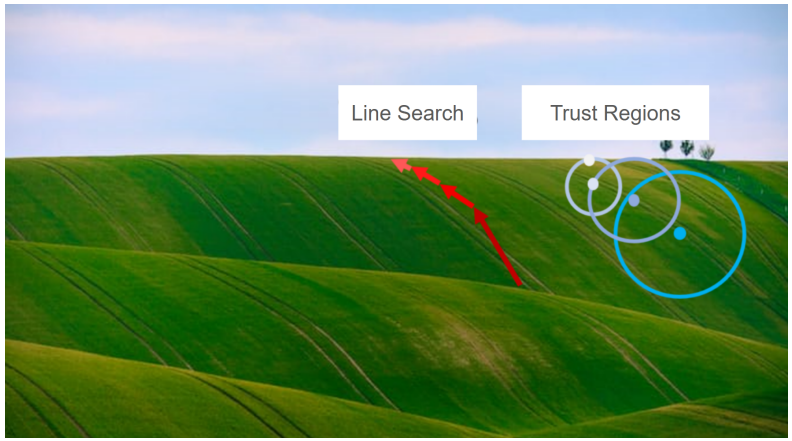


7.5 Policy Optimization

Progresses beyond simple (vanilla) PG such as

- **Trust Region Policy Optimization (TRPO)**
- **Proximal Policy Optimization (PPO):**

On-Policy Optimization : Avoid taking bad actions that collapse the training performance

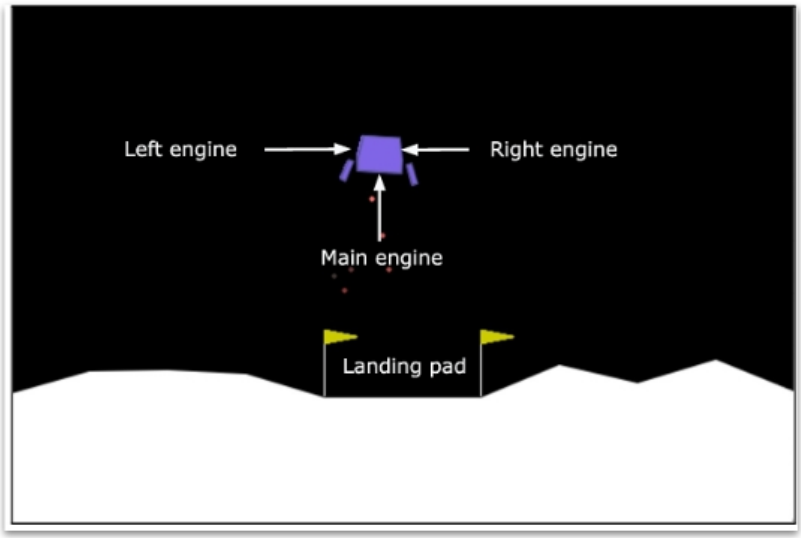


- Line search = First pick action, then step-size (learning rate)
- Trust region = pick learning rate first, then direction

7.6 DRL Tutorial

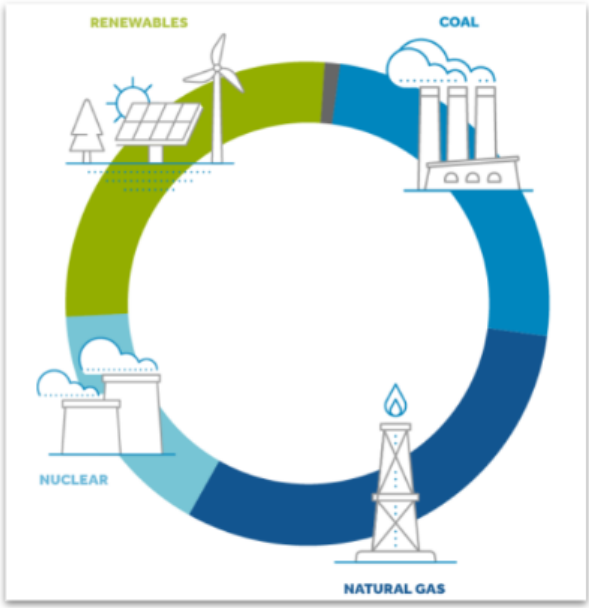
- Two sets of codes. For discrete (Lunar Lander) and continuous (Energy Mix) space

Lunar Lander



Discrete Action Space

Energy mix



Continuous Action Space

- **Discrete action space:** Lunar Lander (OpenAI Gymnasium)
  - Agent learns to land a spacecraft safely using discrete thrust actions.
  - Environment simulates lunar gravity and physics
- **Continuous action space:** Energy Mix Optimization
  - Agent learns to control continuous variables (e.g., energy supply quantities)
  - Demonstrates policy-based DRL using Proximal Policy Optimization (PPO)

7.5 Future of RL