# Reinforcement Learning
# LAB 1

## k-armed Bandits

Alberto Sinigaglia
alberto.sinigaglia@phd.unipd.it

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

# Labs

We will have 4 labs together on RL:

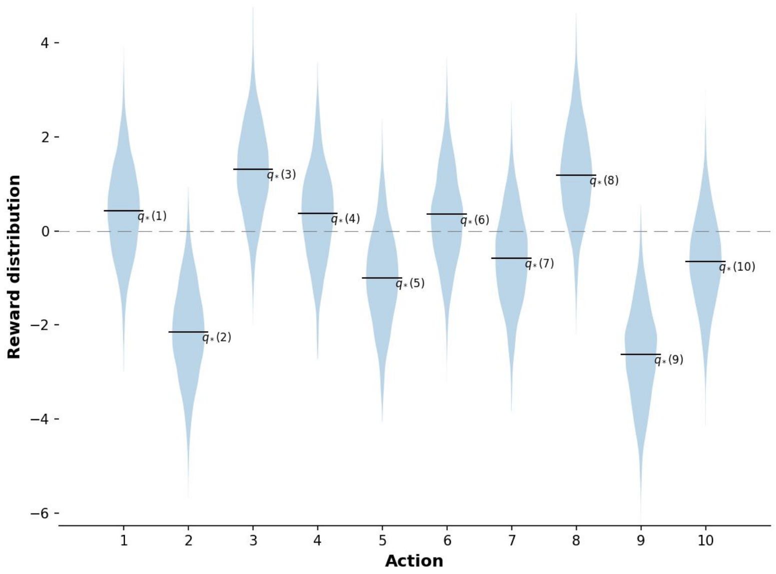
1. today's one on **k-armed bandit**
2. next monday on **Dynamic Programming**
3. the monday after on **Monte Carlo** estimates
4. 13th Nov on **TD learning**

Then 2/4 (depends on you) labs on Deep RL:

5. **intro** to **TF/PyTorch** (probably on separate days)
6. how to use **TF/PyTorch for Deep RL** (probably on separate days)

# Quick recap

1. **K** bandits (slot machines)
2. each bandit has a stochastic reward function
3. we can try them as many times as we want
4. the aim is to find which one is the best, trying to avoid spending billions



# Components

**Policy**

Given an estimate of the $Q$ *function*, which action should I choose?

**Update**

Given an estimate of the $Q$ *function* and a new *reward* $R$, how should I update $Q$?

```
Policy(policy_update, actions, inital_value)
    act() # which action to pick
    step(action, reward) # update belief of Q
    reset() # reset estimates
```

EpsilonGreedyPolicy

UCBPolicy

GradientPolicy

# EPsGreedyPolicy

$$A_t = \begin{cases} argmax_{a \in A}\, q(a) \;\; \text{with } P = 1 - \epsilon \\ \text{uniform random with } P = \epsilon \end{cases}$$

Update:

- Sample average: $$Q_{t+1}(a) = Q_t(a) + \frac{1}{n}(R_t - Q_t(a))$$
- Running average: $$Q_{t+1}(a) = Q_t(a) + \alpha(R_t - Q_t(a))$$

# UCBPolicy

$$A_t = argmax_{a \in A} \left[ Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right]$$

Update:

- Sample average: $$Q_{t+1}(a) = Q_t(a) + \frac{1}{n}(R_t - Q_t(a))$$
- Running average: $$Q_{t+1}(a) = Q_t(a) + \alpha(R_t - Q_t(a))$$

# GradientPolicy

$$P\{A_t = a\} = \frac{e^{H_t(a)}}{\sum_{a'} e^{H_t(a')}}$$

Update:

- Gradient update: $$\begin{cases} H(A_t) = H(A_t) + \alpha(R - \bar{R})(1 - \pi_t(A_t)) \\ H(a) = H(a) - \alpha(R - \bar{R})\pi_t(a), \ \forall a \neq A_t \end{cases}$$

# Why is this formula like this?

$$\begin{cases} H(A_t) = H(A_t) \underline{+} \alpha(R - \bar{R})\underline{(1 - \pi_t(A_t))} \\ H(a) = H(a) \underline{-} \alpha(R - \bar{R})\underline{\pi_t(a)}, \ \forall a \neq A_t \end{cases}$$

# Why is this formula like this?

$$\begin{cases} H(A_t) = H(A_t) \underline{+} \, \alpha(R - \bar{R})\underline{(1 - \pi_t(A_t))} \\ H(a) = H(a) \underline{-} \, \alpha(R - \bar{R})\underline{\pi_t(a)}, \ \forall a \neq A_t \end{cases}$$

$$\begin{cases} H(A_t) = H(A_t) + \alpha(R - \bar{R})(1 - \pi_t(A_t)) \\ H(a) = H(a) + \alpha(R - \bar{R})(0 - \pi_t(a)), \ \forall a \neq A_t \end{cases}$$

# Why is this formula like this?

We can see this as a weighted regression task, where the weight is $(R - \bar{R})$ which, if negative, has a repulsive effect

$$\begin{cases} H(A_t) = H(A_t) \underline{+} \alpha(R - \bar{R})\underline{(1 - \pi_t(A_t))} \\ H(a) = H(a) \underline{-} \alpha(R - \bar{R})\underline{\pi_t(a)}, \ \forall a \neq A_t \end{cases}$$

$$\begin{cases} H(A_t) = H(A_t) + \alpha(R - \bar{R})(1 - \pi_t(A_t)) \\ H(a) = H(a) + \alpha\underbrace{(R - \bar{R})}_{w}\underbrace{(0 - \pi_t(a))}, \ \forall a \neq A_t \end{cases}$$

$$H = H - \alpha \cdot w \cdot \nabla(y - \pi(a))^2$$

$$\frac{\nabla(y - \hat{y})^2}{2} \text{ with } y \in \{0, 1\}, \hat{y} = \pi(a)$$

# Example

Say we have a 3 armed bandits, and we start with a uniform prior over them.
Since GBA uses softmax as transformation of the logits, any uniform initialization will induce a uniform distribution, so say we start with:

$$H(a) = 1, \ \forall a \in A \qquad \qquad \bar{R} = 3$$

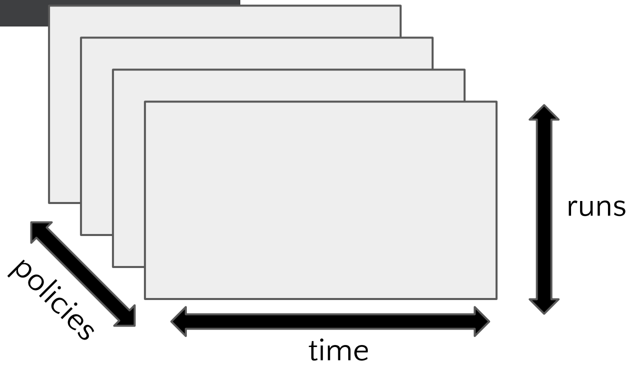
Say we pull the second bandit and we get a reward of 4:

$$\begin{cases} H(A_t) = H(A_t) + \alpha(R - \bar{R})(1 - \pi_t(A_t)) \\ H(a) = H(a) - \alpha(R - \bar{R})\pi_t(a), \ \forall a \neq A_t \end{cases} \Rightarrow \begin{cases} H(A_t) = H(A_t) + \alpha(4 - 3)(1 - 0.\bar{3}) \\ H(a) = \ H(a) + \ \alpha(4 - 3)(0 - 0.\bar{3}), \ \forall a \neq A_t \end{cases}$$

$$\begin{cases} H(A_t) = H(A_t) + \alpha \cdot 0.\bar{6} \\ H(a) = \ H(a) \ - \alpha \cdot 0.\bar{3}, \ \forall a \neq A_t \end{cases}$$

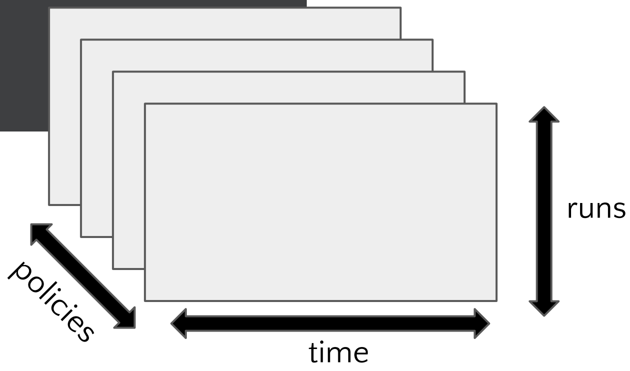
# Simulation

```python
def simulate(runs, time, policies: List[Policy], environment: Environment):
    rewards = np.zeros((len(policies), runs, time))
    best_action_counts = np.zeros(rewards.shape)
    for i, policy in enumerate(policies):
        for r in trange(runs):
            policy.reset()
            environment.reset()
```



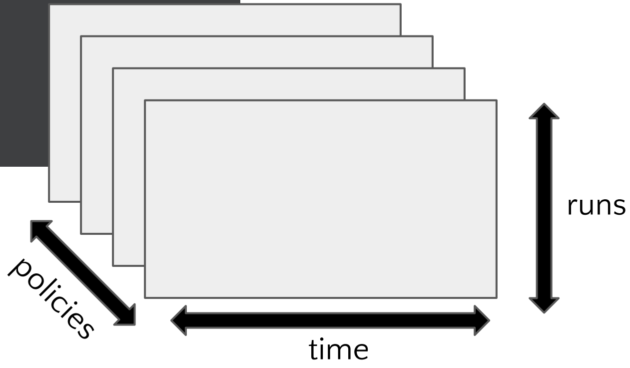
# Simulation

```python
def simulate(runs, time, policies: List[Policy], environment: Environment):
    rewards = np.zeros((len(policies), runs, time))
    best_action_counts = np.zeros(rewards.shape)
    for i, policy in enumerate(policies):
        for r in trange(runs):
            policy.reset()
            environment.reset()
            for t in range(time):
                action = policy.act()
                reward = environment.reward(action)
                policy.step(action, reward)
                rewards[i, r, t] = reward
```



# Simulation

```python
def simulate(runs, time, policies: List[Policy], environment: Environment):
    rewards = np.zeros((len(policies), runs, time))
    best_action_counts = np.zeros(rewards.shape)
    for i, policy in enumerate(policies):
        for r in trange(runs):
            policy.reset()
            environment.reset()
            for t in range(time):
                action = policy.act()
                reward = environment.reward(action)
                policy.step(action, reward)
                rewards[i, r, t] = reward
                if action == environment.best_action:
                    best_action_counts[i, r, t] = 1
```



# Simulation

```python
def simulate(runs, time, policies: List[Policy], environment: Environment):
    rewards = np.zeros((len(policies), runs, time))
    best_action_counts = np.zeros(rewards.shape)
    for i, policy in enumerate(policies):
        for r in trange(runs):
            policy.reset()
            environment.reset()
            for t in range(time):
                action = policy.act()
                reward = environment.reward(action)
                policy.step(action, reward)
                rewards[i, r, t] = reward
                if action == environment.best_action:
                    best_action_counts[i, r, t] = 1
    mean_best_action_counts = best_action_counts.mean(axis=1)
    mean_rewards = rewards.mean(axis=1)
    return mean_best_action_counts, mean_rewards
```