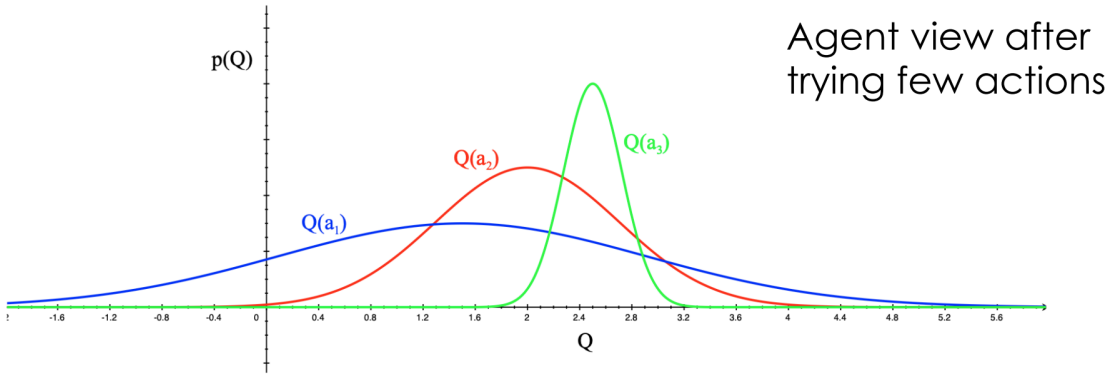
Nel Reinforcement Learning l'approccio utilizzato per la scelta delle azioni può essere:

- *Naive Exploration* $\longrightarrow$ Aggiunge rumore alla *policy* greedy (ad es. $\epsilon-$greedy)
- *Optimistic Initialisation* $\longrightarrow$ Assume il meglio finché non viene provato diversamente
- *Optimistim in the Face of Uncertinty* $\longrightarrow$ Preferisce le azioni con valori incerti.

Incertezza

C'è sempre una incertezza *intrinseca* nell'accuratezza della nostra stima.

- *Problema facile*: ci sono due azioni, una delle due è sempre buona mentre l'altra è sempre cattiva: una volta provate entrambe abbiamo finito. (In quanto sceglieremo poi sempre la migliore)
- *Problema difficile*: ci sono due azioni, una è molto meglio dell'altra ma c'è tanto **rumore** e ciò richiede parecchio tempo per disambiguare (e capire quindi che un'azione è molto migliore dell'altra).



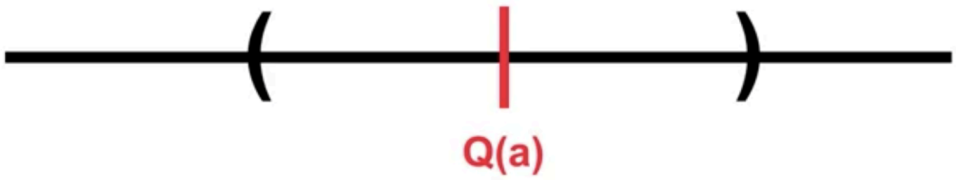
- Which action should we pick?
- The more uncertain we are about an action-value
- The more important it is to explore that action
- It could turn out to be the best action

Il principio dell'**optimism in the face of uncertainty** dice: "non prendere l'azione che credi migliore, ma prendi l'azione che potenzialmente può fare meglio di tutte".

- Più siamo incerti nei confronti di un'azione e più dobbiamo *esplorare* questa azione.
- Come possiamo **stimare** l'incertezza?
- Come possiamo **ridurre** l'incertezza?

Incertezza nella stima di Q

- What does it mean to have uncertainty in the estimates?



- Q(a) represents our current estimate for action a.
- The brackets represent a confidence interval around q*(a)
- Brackets say we are **confident that the value of action a lies somewhere in this region**

- La regione tra le parentesi è l'intervallo di confidenza il quale rappresenta la nostra incertezza ⟶ Più questo intervallo è **piccolo** e **più siamo certi** della nostra stima, al contrario più questo intervallo è **grande** e **più siamo incerti**.

Upper Confidence Bound (UCB)

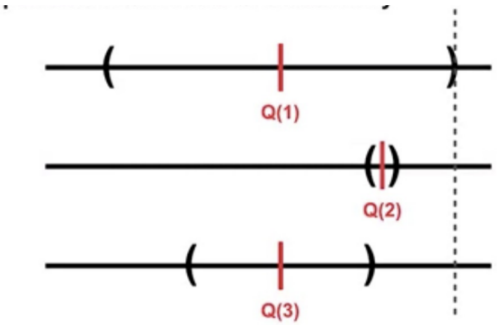
UCB segue il principio dell'*optimism in the face of uncertainty*:

> Se siamo **incerti** di qualcosa allora dovremmo ottimisticamente assumere che sia buona.

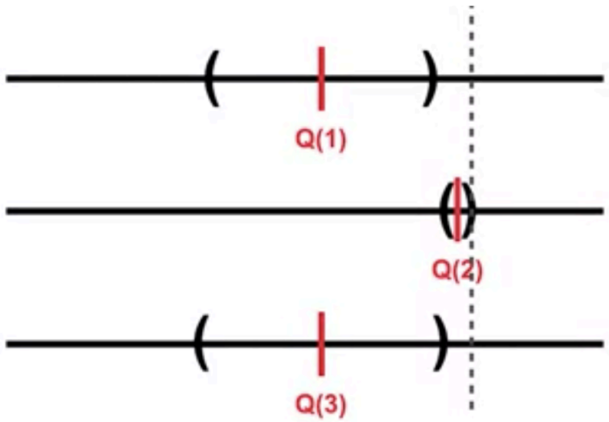
UCB: example

- We have three actions with associated uncertainties,
- Our agent has no idea which is best
- So it optimistically picks the action that has the highest upper bound
  - It does have the highest value and we get good reward

  OR
  - we get to learn about an action we know least about like the example on the slide.



- Let's let the algorithm pick one more action.
- This time Q2 has the highest upper-confidence bound because it's estimated value is highest, even though the interval is small



UCB's action selection

$$A_t \doteq \arg\max_a \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right]$$

exploitation

exploration

- We will select the action that has the **highest estimated value plus the upper-confidence bound exploration term**.
- The C parameter is a user-specified parameter that controls the amount of exploration

  - Dove $N_t(a)$ rappresenta il numero di volte, allo step $t-$esimo in cui l'azione a è stata presa.
  - Il termine in azzurro rappresenta l'incertezza

Esempio

- 10.000 steps so far
- Imagine we have selected action a 5,000 times, then the uncertainty term here will be (0.043 * c)
- If instead we had only selected action a 100 times, the uncertainty term would be 10 times larger.

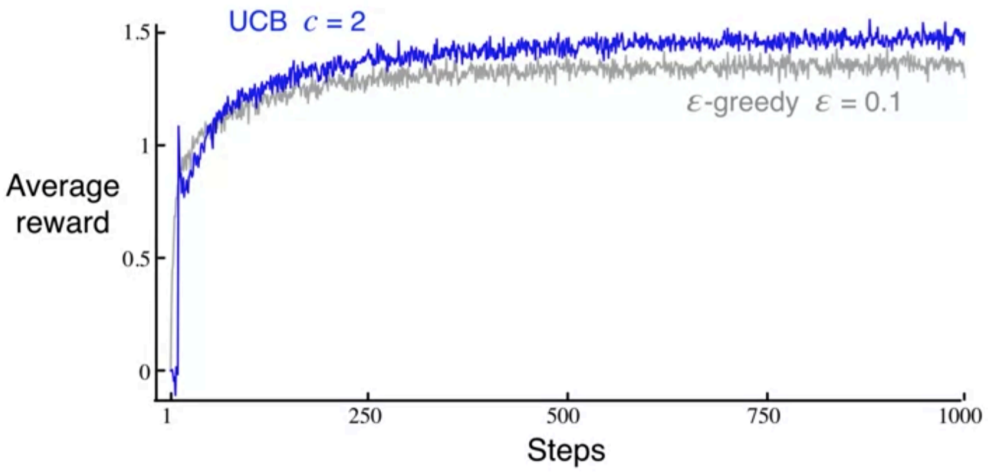
$$c\sqrt{\frac{ln\ t}{N_t(a)}} \rightarrow c\sqrt{\frac{ln\ \text{timesteps}}{\text{times action } a \text{ taken}}}$$

$$c\sqrt{\frac{ln\ 10000}{5000}} \rightarrow 0.043c$$

$$c\sqrt{\frac{ln\ 10000}{100}} \rightarrow 0.303c$$

UCB performance

- Performance of optimistic initial values



- Initially, UCB explores more to systematically reduce uncertainty
- UCB's exploration reduces over time whereas Epsilon-greedy continues to take a random action 10 percent of the time

In definitiva *UCB* ha buone performance ma ha delle difficoltà nel trattare i problemi *nonstazionari*.

Un metodo alternativo: La preferenza

I metodi visti finora erano basati sulla stima degli *action-value* e l'uso di queste stime viene quindi usato epr scegliere le azioni.
L'**alternativa** è imparare una preferenza numerica $H_t(a)$ per ogni azione a; maggiore sarà la preferenza e più spesso l'azione viene presa.
Problema: la *preferenza* non ha interpretazione in termini di *reward*.

Softmax function

- Trasforma un vettore di K valori reali in un vettore di K valori che se sommati danno come risultato 1. In pratica trasforma un vettore in input in una probabilità.

$$S(\mathbf{a}): \begin{bmatrix} a_1 \\ a_2 \\ \dots \\ a_N \end{bmatrix} \rightarrow \begin{bmatrix} S_1 \\ S_2 \\ \dots \\ S_N \end{bmatrix}$$

$$S_j = \frac{e^{a_j}}{\sum_{k=1}^{N} e^{a_k}} \qquad \forall j \in 1..N$$

Action preference

L'idea è quella di considerare la preferenza relativa di un'azione su un'altra.
Le *action probabilities* sono determinate secondo una **distribuzione soft-max**:

$$Pr\{A_t = a\} = \frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}} = \pi_t(a)$$

Dove $\pi_t(a)$ è la *probabilità di prendere l'azione* a *allo step* t.
Inizialmente $H_1(a) = 0$ per ogni a, quindi tutte le azioni hanno la stessa probabilità di essere selezionate.

Gradient Bandit algorithm

Ad ogni step dopo aver scelto l'azione A_t e aver ricevuto la *reward* R_t la *action preference* viene aggiornata come segue:

$$\begin{aligned} H_{t+1}(A_t) &\doteq H_t(A_t) + \alpha\big(R_t - \bar{R}_t\big)\big(1 - \pi_t(A_t)\big), && \text{and} \\ H_{t+1}(a) &\doteq H_t(a) - \alpha\big(R_t - \bar{R}_t\big)\pi_t(a), && \text{for all } a \neq A_t, \end{aligned}$$

Dove $\alpha > 0$ è il *parametro* della step-size e $\overline{R} \in \mathbf{R}$ è la media di tutte le reward fino all'istante t, incluso. $\overline{R}$ agisce come *baseline*, cioè:

- Se la *reward* $>$ *baseline* $\longrightarrow$ la probabilità di prendere l'azione A_t nel futuro cresce.
- Se la *reward* $>$ *baseline* $\longrightarrow$ la probabilità di prendere l'azione A_t decresce.

Regret

Utiliziamo questo strumento per valutare gli algoritmi in questo contesto, ovvero per capire *quanto peggio* facciamo rispetto al valore ottimale.

- L'action-value è la media delle reward per l'azione a $\longrightarrow\ Q(a) = \mathbf{E}[r|a]$
- L'**optimal-value** $V* = Q(a*) = \underset{a \in \mathbf{A}}{max}\ Q(a)$
- La **regret** è la perdita di un'opportunita per uno step $\longrightarrow\ l_t = \mathbf{E}[V* - Q(a_t)]$
- La **total regret** è la perdita totale di opportunità $\longrightarrow\ L_t = \mathbf{E}\left[\sum_{\pi=1}^{t} V* - Q(a_\pi)\right]$
- *Maximize cumulative reward = minimize total regret*

Counting regret

- Il contatore $N_t(a)$ è il numero di selezioni per l'azione a
- $\Delta a = V* - Q(a)$, ovvero il gap tra il valore dell'azione a è l'azione ottima $a*$
- Regret è una funzione di *gap* e *contatori*:

$$
\begin{aligned}
L_t &= \mathbb{E}\left[\sum_{\tau=1}^{t} V^* - Q(a_\tau)\right] \\
&= \sum_{a \in \mathcal{A}} \mathbb{E}\left[N_t(a)\right](V^* - Q(a)) \\
&= \sum_{a \in \mathcal{A}} \mathbb{E}\left[N_t(a)\right] \Delta_a
\end{aligned}
$$

- Un buon algoritmo assicura piccoli contatori per grandi gap.
- Problema: **I gap sono sconsciuti!!!**

Contextual bandit

- Trovare la migliore azione quando il task è stazionario, e
- Tracciare come cambia nel tempo l'azione migliore quando il task è non stazionario.
- Imparare una mappa: *situazioni* $\longrightarrow$ *azioni migliori nella situazione data*
- Possiamo usare il contesto per scegliere le azioni.
- Potrebbe esistere una buona **policy** (decision rule) per scegliere le azioni basandosi sul contesto. Ad es. Se maschio $\longrightarrow$ scegli azione 2 altrimenti se ha un età>45 scegli azione 1... E così via.

Multi-armed bandit

- Nessun contesto
- Provare a fare così come migliroe azione singola (es. un singolo trattamento giusto per tutta la popolazione).