# Artificial Intelligence

Augustin Cosse.


Fall 2020

November 16, 2020

# So far

- Simple reflex, random agents, Utility based, Goal based
- Improvement through Search Methods (uninformed (DFS, BFS), informed (BS, A*)).
- Logical Reasoning, Propositional logic + First Order Logic, Inference
- Learning
  - Decision trees, regression, classification
  - Neural Networks
  - Parametric vs non parametric
  - Kernels and SVMs
  - Unsupervised and Clustering
  - Reinforcement learning

# Parametric vs Non Parametric

- Linear regression, logistic regression and neural networks use the training data to estimate a fixed set of parameters
- Those parameters define our hypothesis $h_{\beta}(\boldsymbol{x})$. Once we have the hypothesis, we can just throw away the training data
- A learning model that summarizes data with a set of parameters of fixed size is called a parametric model
- No matter how much data you throw at a parametric model, it won't change its mind about how many parameters it needs
- A non parametric model is one that cannot be characterized by a bounded set of parameters

# Parametric vs Non Parametric

- A example of a non parametric model for classification is the K nearest neighbor (KNN) classifier
- Given a query $\boldsymbol{x}_q$, KNN works by first finding the $k$ examples that are the nearest to $\boldsymbol{x}_q$
- In classification, we then simply take the majority vote across the neighbors
- In regression, we can take the mean, or median of the neighbors, or solve a regression problem on the neighbors

# Parametric vs Non Parametric

- Another popular non parametric approach in classification are Support Vector Machines (or Max Margin Classifiers).
- SVMs is currently the most popular approach for 'off the shelf' supervised learning. If you don't have any specialized prior knowledge about a domain, the SVM is an excellent method to try first

# Parametric vs Non Parametric

- There are three properties that make SVM attractive:
  - SVM constructs a maximum margin separator (decision boundary with the largest possible distance to example points)
  - SVM creates a linear separating plane but they have the ability to embed the data into a higher dimensional space using the so called kernel trick
  - SVM are a nonparametric method (they retain training examples and potentially need to store them all). However in practice they often end up retaining only a small fraction of the examples. Thus they combine the advantages of nonparametric and parametric models: they have the flexibility to represent complex functions but they are resistant to overfitting.

# SVM

- The general form of an SVM is the following

$$h(\boldsymbol{x}) = \text{sign}\left(\sum_{i \in \mathcal{D}} \alpha_i t^{(i)} \langle \boldsymbol{x}^{(i)}, \boldsymbol{x} \rangle - b\right)$$

  where the $\alpha_i$ are non zero only for the support vectors (examples lying on the margins)

- The model above is still a linear model (i.e. we have a linear combination of the features $\boldsymbol{x}_j^{(i)}$ and we take the sign of this combination)

- What if the data is not linearly separable?

# Kernels

- In linear regression we have seen that we could generate higher dimensional feature vectors $\phi(\boldsymbol{x}^{(i)})$ to replace $\boldsymbol{x}^{(i)}$
- We can then substitute those vectors to get the expression

$$h(\boldsymbol{x}) = \text{sign}\left(\sum_{i\in\mathcal{D}} \alpha_i t^{(i)} \langle \phi(\boldsymbol{x}^{(i)}), \phi(\boldsymbol{x})\rangle - b\right)$$

- It turns out that the inner product $\langle \phi(\boldsymbol{x}^{(i)}), \phi(\boldsymbol{x})\rangle$ can often be computed without computing the feature vectors explicitely.
- Instead of thinking in terms of feature vectors, we can think in terms of similarity and replace the inner product by a similarity function whcih we call kernel

$$h(\boldsymbol{x}) = \text{sign}\left(\sum_{i\in\mathcal{D}} \alpha_i t^{(i)} \kappa(\boldsymbol{x}^{(i)}, \boldsymbol{x}) - b\right)$$

# Kernels

- The most popular example of such function is the Gaussian kernel

$$\kappa(\boldsymbol{x}, \boldsymbol{x}') = \exp(-\frac{\|\boldsymbol{x} - \boldsymbol{x}'\|^2}{\sigma})$$

- Just as the inner product, you see that $\kappa(\boldsymbol{x}, \boldsymbol{x}')$ will be larger when $\boldsymbol{x}$ is similar to $\boldsymbol{x}'$

- Learning a classifier with a Gaussian kernel corresponds to centering a Gaussian with a particular width around each sample, and weighting that Gaussian by the target

# Kernel

- The most popular example of such function is the Gaussian kernel

$$\kappa(\boldsymbol{x}, \boldsymbol{x}') = \exp(-\frac{\|\boldsymbol{x} - \boldsymbol{x}'\|^2}{\sigma})$$

- Just as the inner product, you see that $\kappa(\boldsymbol{x}, \boldsymbol{x}')$ will be larger when $\boldsymbol{x}$ is similar to $\boldsymbol{x}'$

- Learning a classifier with a Gaussian kernel corresponds to centering a Gaussian with a particular width around each sample, and weighting that Gaussian by the target

# Unsupervised Learning

- The most popular family of unsupervised algorithms are the clustering algorithms
- Among those algorithms, the most commonly used is K-means.
- In K-means the algorithm iterates between the following two steps:
  - Compute the center of mass of each cluster and define the centroids as those centers
  - Update the assignment by associating each point to its nearest centroid

# Reinforcement learning

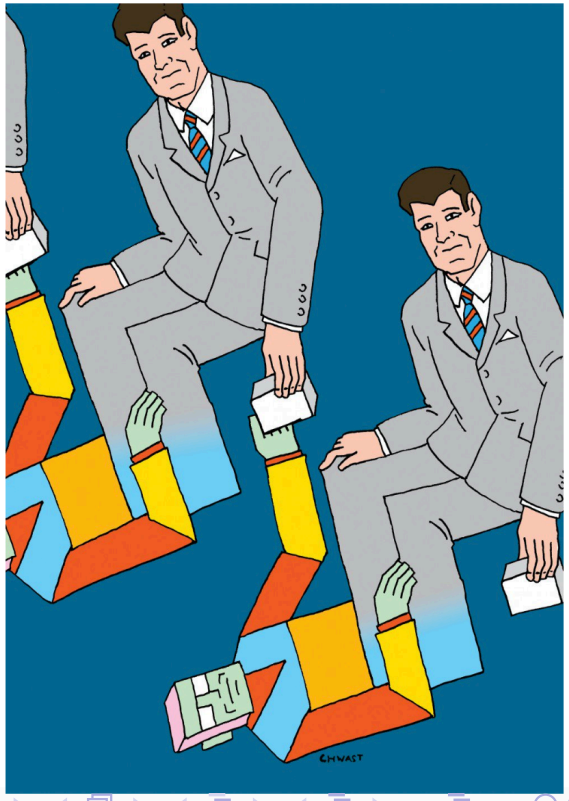
- Reinforcement learning is learning what to do so as to maximize a numerical reward signal
- "The learner is not told which action to take but instead must discover which action yield the most reward by trying them"
- Ex.1.: "A chess player makes a move. The choice is informed by planning (anticipation of possible replies and counterreplies) and by immediate intuitive judgements of the desirability of possible positions and moves"
- Ex.2. "A mobile robot decides whether it should enter a room in search of a target or start to find its way back to its battery charging station"

source: R. Sutton, A.G. Barto, Reinforcement learning: An introduction

# 10 BREAKTHROUGH TECHNOLOGIES

## Reinforcement Learning

**By experimenting, computers are figuring out how to do things that no programmer could teach them.**



**Availability: 1 to 2 years**

by Will Knight

# Reinforcement learning: constitutive elements

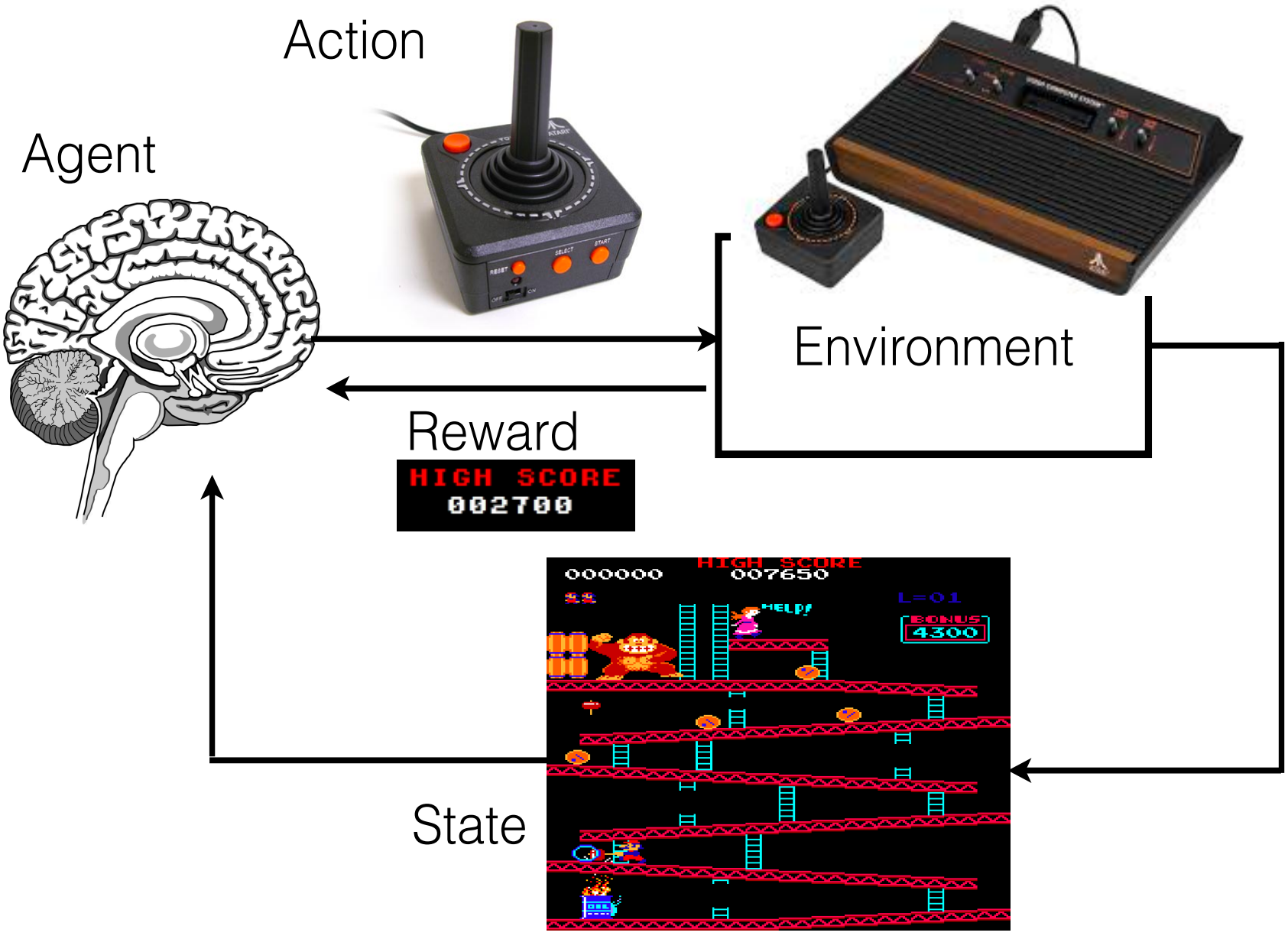
- The policy defines the learning agent's way of behaving at any given time
- On each time step, the evironment sends to the reinforcement learning agent a single number called the reward which specifies what are good and bad events in an immediate sense.
- To know what is good in the long run, we use a value function which is the total amount of reward the agent can expect to accumulate over the future, starting from that state.
- Finally, the last element is a model for the environment which enables inferences to be made on how the environment will react w.r.t a particular action. The role of the model is essentially to predict the next state and next reward given the current state and action.

source: R. Sutton, A.G. Barto, Reinforcement learning: An introduction

# Reinforcement learning

- Action are usually taken to maximize the value, not the reward, because high value actions are those that lead to the highest level of reward in the long run. Finding such actions is however hard as those have to be constantly re-estimated based on the decisions of the agent.
- An important instance of reinforcement learning in which there is a single state is the bandit problem

source: R. Sutton, A.G. Barto, Reinforcement learning: An introduction

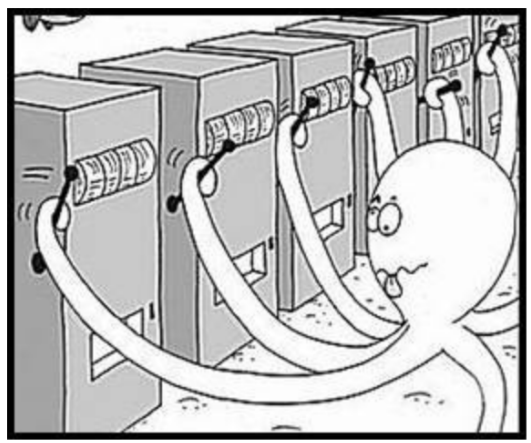



Inspired from https://keon.io/deep-q-learning/, Deep Q-Learning with Keras and Gym

# Reinforcement learning

- The multi-armed bandit is a simplified version of non associative feedback problem

- In the k-armed bandit problem, you are faced repeatedly with a choice among *k* different possible options or actions. After each choice, you receive a numerical reward chosen from some stationnary probability distribution that depends on the action you selected.



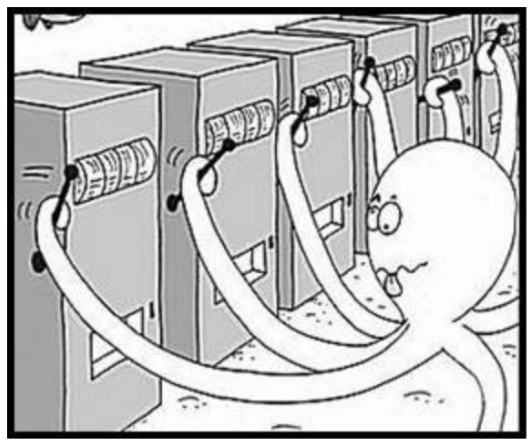
source: Sutton & Barto, Reinforcement Learning: An Introduction.

# Reinforcement learning

- You can think of the $k$-armed bandit problem as the problem of playing one of the $k$ levers of a slot machine. You choose which lever you play and the reward is the payoff for hitting the jackpot

- The value of an arbitrary action, $a$, which we denote $v(a)$ is the expected reward given that you selected $a$

$$v(a) = \mathbb{E}\{R_t | A_t = a\}$$



source: Sutton & Barto, Reinforcement Learning: An Introduction.

# Reinforcement learning

- We don't know the exact value $v^*(a)$ (because we don't know the distribution). So we would like an estimate $v_{\text{est},t}(a)$ (estimated value at time $t$) that would be as close as possible to $v^*(a)$
- When you keep track of the estimated action values through time, then at each time step, there is always at least one action whose estimated value is best. We call this greedy actions.
- When you select one of these actions, we say that you are exploiting your current knowledge of the values of the actions
- When you select one of the non greedy actions, then we say you are exploring. In particular, exploring enables you to improve your estimates of the non greedy action's values.

source: Sutton & Barto, Reinforcement Learning: An Introduction.

# Reinforcement learning

- Exploitation will maximize your expected reward on the one step but exploration may lead to greater total reward in the long run.
- Intuitively, if you have many time steps ahead, it may be better to explore.
- How do we balance exploration and exploitation when dealing with the $k$-armed bandit problem?

source: Sutton & Barto, Reinforcement Learning: An Introduction.

# Reinforcement learning: Action value estimates

- The first thing we want to do is get an estimate of the value of an action at time $t$.
- The natural approach is to average over the rewards received in the past

$$v_{\text{est},t}(a) = \frac{\text{sum of rewards when } a \text{ taken}}{\text{number of times } a \text{ taken}} = \frac{\sum_{i=1}^{t-1} R_i \mathbb{1}_{A_i=a}}{\sum_{i=1}^{t} \mathbb{1}_{A_i=a}}$$

Here $\mathbb{1}_{\text{predicate}}$ is used to denote the indicator function for the predicate. $\mathbb{1}_p = 1$ if the predicate is verified and 0 otherwise.

source: Sutton & Barto, Reinforcement Learning: An Introduction.

# Reinforcement learning: Action value estimates

- Then the simplest action selection procedure (known as greedy action selection) is to select (one of) the action(s) with the highest estimated value,

$$A^* = \underset{a}{\text{argmax}}\ v_{\text{est},t}(a)$$

- Greedy action selection always exploits current knowledge to maximize immediate reward (i.e it does not spend time investigating inferior actions to see if they might be better)

- A group of alternative methods known as $\varepsilon$-greedy methods consist in behaving greedily most of the time, but once in a while (with probability $\varepsilon$) select an action randonly (with uniform probability) from the list of all possible actions.

source: Sutton & Barto, Reinforcement Learning: An Introduction.

# Reinforcement learning: Action value estimates

- $\varepsilon$-greedy methods ensure that every action is sampled an infinite number of times. Which in turns implies that the estimator $v_{\text{est}}(a)$ converges to the $v^*$ (the true expected value)

- To avoid keeping each reward in memory independently, typical implementations of greedy and $\varepsilon$-greedy only update the averaged reward (a.k.a value). If $Q_n$ is used to denote the value of a given action after the $n^{th}$ step,

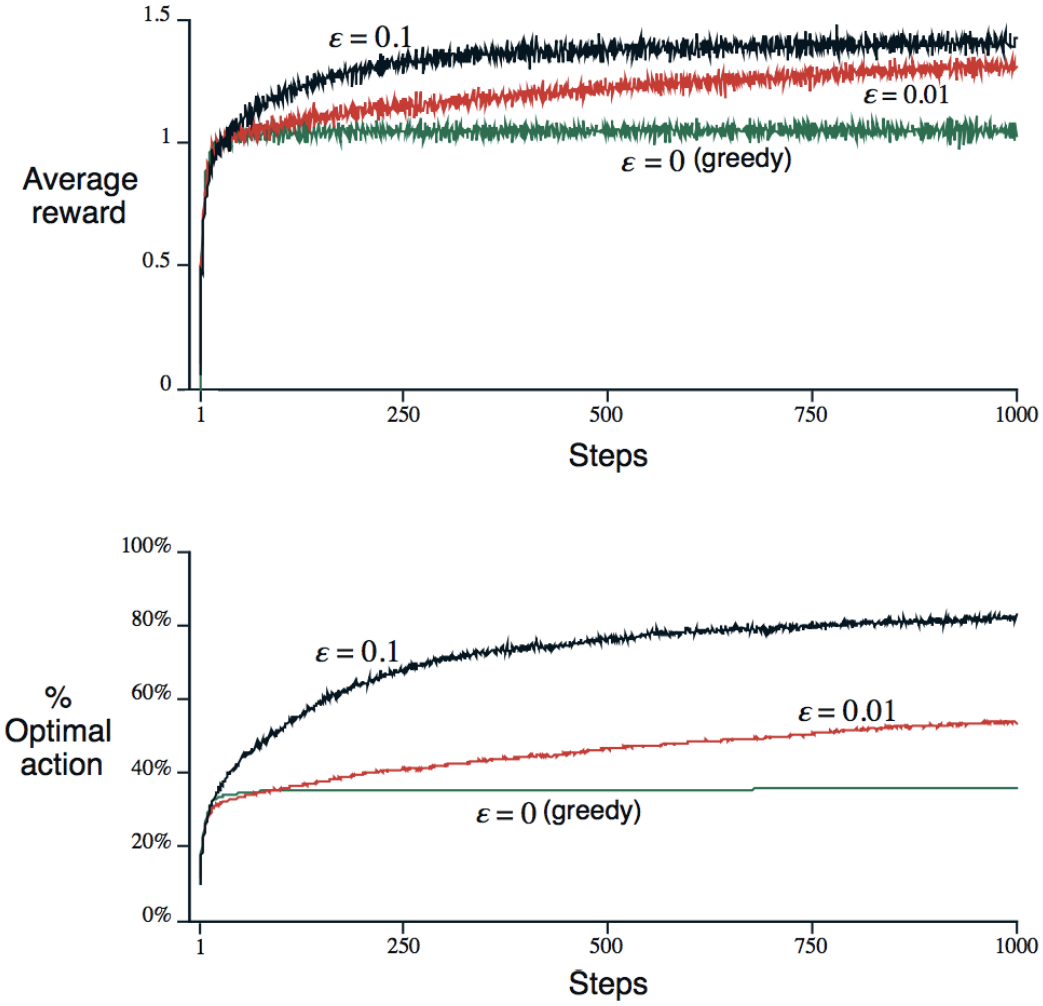
$$Q_n = \frac{R_1 + R_2 + \ldots R_{n-1}}{n-1}$$

we compute $Q_{n+1}$ as

$$\begin{aligned} Q_{n+1} = \frac{1}{n}\sum_{i=1}^{n} R_i &= \frac{1}{n}\left(R_n + (n-1)\frac{1}{n-1}\sum_{i=1}^{n-1} R_i\right) \\ &= Q_n + \frac{1}{n}\left[R_n - Q_n\right] \end{aligned}$$

source: Sutton & Barto, Reinforcement Learning: An Introduction.

# Reinforcement learning: greedy vs $\varepsilon$-greedy




source: Sutton & Barto, Reinforcement Learning: An Introduction.

# Reinforcement learning: Simple Bandit algorithm

1. Initialize, for every action $a = 1$, to $k$

   1.1 $v(a) \leftarrow 0$

   1.2 $n(A) \leftarrow 0$ (number of times $A$ has been chosen)

2. Repeat

   2.1 $A \leftarrow \begin{cases} \underset{a}{\text{argmax}}\ v(a) & \text{with probability } 1-\varepsilon \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$

   2.2 $R \leftarrow \text{bandit}(a)$

   2.3 $n(A) \leftarrow n(A) + 1$

   2.4 $q(A) \leftarrow v(A) + \frac{1}{N(A)}[R - v(A)]$

source: Sutton & Barto, Reinforcement Learning: An Introduction.

# Reinforcement learning: Simple Bandit algorithm

- So far we have focused on stationnary Bandit problems (Problems for which the reward probabilities do not change with time).

- When the problems are not stationnary, the choice of an action will depend on the instant at which the action is taken. In particular we will want to give more weight to the rewards associated to more recent actions. One way to achieve this is to add a weight in the update rule for the value $V_{n+1}$,

$$v_{n+1} = v_n + \alpha(R_n - v_n)$$

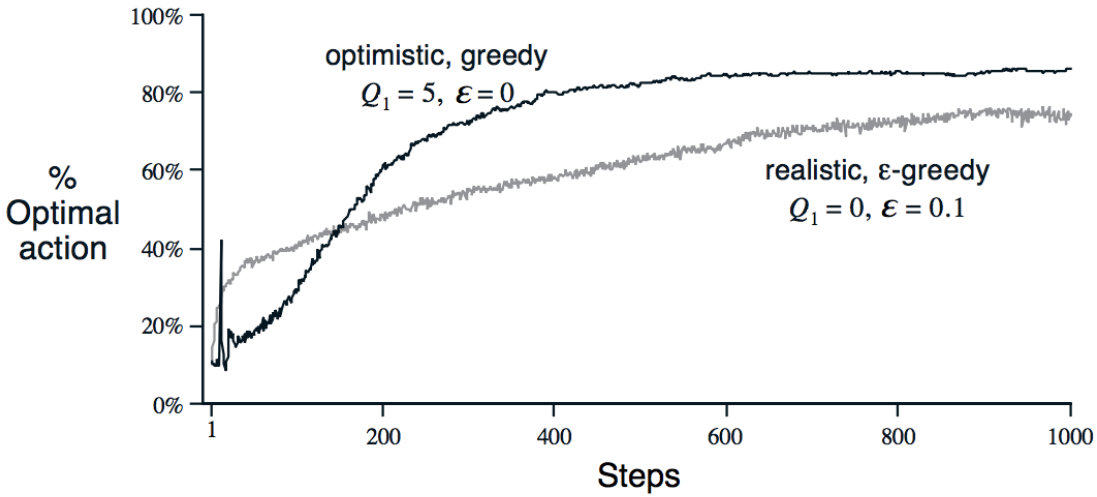
Developing, we get

$$v_{n+1} = (1-\alpha)^n v_1 + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} R_i$$

source: Sutton & Barto, Reinforcement Learning: An Introduction.

# Reinforcement learning: Simple Bandit algorithm

- If $\alpha = 1$, $1 - \alpha = 0$ and all the weight goes to the very last reward
- This idea of associating high initial values to every action in order to force an initial decrease of the greedy search method is known as optimistic initial values.




source: Sutton & Barto, Reinforcement Learning: An Introduction.

# Policy gradient/gradient Bandit algorithm

- Instead of taking the action that maximizes the value at each step (as in the greedy approach), one can instead introduce policies (i.e. probability that one action is optimal against the others)

- In Gradient bandit algorithms, we define policies (and the corresponding preferences $H_t$) by means of a softmax distribution (equiv. Gibbs or Boltzmann distribution),

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

  here $\Pr\{A_t = a\}$ really means "the probability that the optimal action at time $t$ is $a$".

source: Sutton & Barto, Reinforcement Learning: An Introduction.

# Policy gradient/gradient Bandit algorithm

- Policy gradient algorithms then increase the preference of an action when the reward associated to this action is larger than a baseline ($\overline{R}_t$) which is the average of all previous rewards.
- All the other actions are updated in the opposite direction

$$H_{t+1}(A_t) \leftarrow H_t(A_t) + \alpha(R_t - \overline{R}_t)(1 - \pi_t(A_t))$$
$$H_{t+1}(a) \leftarrow H_t(a) - \alpha(R_t - \overline{R}_t)\pi_t(a), \quad \text{for all } a \neq A_t$$

- We compare the current reward for action $a$ to the average reward $\overline{R}_t(a)$. If $\overline{R}_t > R_t$, the agent interpret the action as being suboptimal compared to previous ones and hence reduces its weight more.
- The weighting by the policies $\pi_t$ has a similar interpretation.

sources: Sutton & Barto, Reinforcement Learning: An Introduction.

# Reinforcement learning: Associative search (contextual bandit)

$$H_{t+1}(A_t) \leftarrow H_t(A_t) + \alpha(R_t - \overline{R}_t)(1 - \pi_t(A_t))$$
$$H_{t+1}(a) \leftarrow H_t(a) - \alpha(R_t - \overline{R}_t)\pi_t(a), \quad \text{for all } a \neq A_t$$

- When an action has a low probability of being selected, there is no need to decrease the weight of this action anymore as it cannot really be held responsible for the fact that the average reward is lower than the current reward.
- On the opposite, if one action $a$ has a relatively higher probability of being selected but is different from the current optimal action $A_t$, it probably contributed for most of the (underoptimal) average $\overline{R}_t$ and should be considered as suboptimal.

source: Sutton & Barto, Reinforcement Learning: An Introduction.

# Bandit/Policy gradient as stochastic gradient ascent

- The Bandit gradient method or policy gradient method has an interpretation as stochastic gradient ascent
- To see this, note that to update the preferences, we will want to follow the direction that maximizes the average reward,

$$H_{t+1}(a) \leftarrow H_t(a) + \alpha \frac{\partial \mathbb{E}[R_t]}{\partial H_t(a)}$$

Here $\mathbb{E}\{R_t\}$ is viewed as a multivariate function in the preferences.

source: Sutton & Barto, Reinforcement Learning: An Introduction.

# Bandit/Policy gradient as stochastic gradient ascent

- In practice, we do not know the population average $\mathbb{E}[R_t] = \sum_b \pi_t(b) v_*(b)$ but let us forget that for the moment

$$\begin{aligned}
\frac{\partial \mathbb{E}[R_t]}{\partial H_t(a)} &= \frac{\partial}{\partial H_t(a)} \left[ \sum_b \pi_t(b) v_*(b) \right] \\
&= \sum_b v_*(b) \frac{\partial \pi_t(b)}{\partial H_t(a)} \\
&= \sum_b (v_*(b) - X_t) \frac{\partial \pi_t(b)}{\partial H_t(a)}
\end{aligned}$$

  The last line follows from the definition of the policy and the fact that $\sum_b \frac{\partial \pi_t(b)}{\partial H_t(a)} = 0$

source: Sutton & Barto, Reinforcement Learning: An Introduction.

# Bandit/Policy gradient as stochastic gradient ascent

- Developing further, we get

$$\begin{aligned}
\frac{\partial \mathbb{E}[R_t]}{\partial H_t(a)} &= \frac{\partial}{\partial H_t(a)} \left[ \sum_b \pi_t(b) v_*(b) \right] \\
&= \sum_b \pi_t(b)(v_*(b) - X_t) \frac{\partial \pi_t(b)}{\partial H_t(a)} / \pi_t(b) \\
&= \mathbb{E}_{A_t} \left\{ (v_*(A_t) - X_t) \frac{\partial \pi_t(A_t)}{\partial H_t(a)} / \pi_t(A_t) \right\} \\
&= \mathbb{E}_{A_t} \left\{ (R_t - \overline{R}_t) \frac{\partial \pi_t(A_t)}{\partial H_t(a)} / \pi_t(A_t) \right\}
\end{aligned}$$

The last line follows from the definition of the value $v_*(A_t)$ of an action $A_t$ as the average reward $\mathbb{E}[R_t]$.

source: Sutton & Barto, Reinforcement Learning: An Introduction.

# Bandit/Policy gradient as stochastic gradient ascent

- If we assume for now that

$$\frac{\partial \pi_t(A_t)}{\partial H_t(a)} / \pi_t(A_t) = \pi_t(A_t)(\mathbb{I}_{a=A_t} - \pi_t(A_t)) \quad ,$$

we get

$$\begin{aligned} \frac{\partial \mathbb{E}[R_t]}{\partial H_t(a)} &= \mathbb{E}\left[(R_t - \overline{R}_t)\pi - t(A_t)(\mathbb{I}_{a=A_t} - \pi_t(a))/\pi_t(A_t)\right] \\ &= \mathbb{E}\left[(R_t - \overline{R}_t)(\mathbb{I}_{a=A_t} - \pi_t(a))\right] \end{aligned}$$

- Stochastic gradient is used to maximize (resp. minimize) a population average by replacing this population average with a sample average ($\mathbb{E}V \to \sum_i V_i$). In its most compressed version, it actually defines the iterates by taking one sample at a time.

source: Sutton & Barto, Reinforcement Learning: An Introduction.

# Bandit/Policy gradient as stochastic gradient ascent

- In this framework, the gradient

$$\frac{\partial \mathbb{E}[R_t]}{\partial H_t(a)} = \mathbb{E}\left[(R_t - \overline{R}_t)(\mathbb{1}_{a=A_t} - \pi_t(a))\right]$$

  is turned into the updates

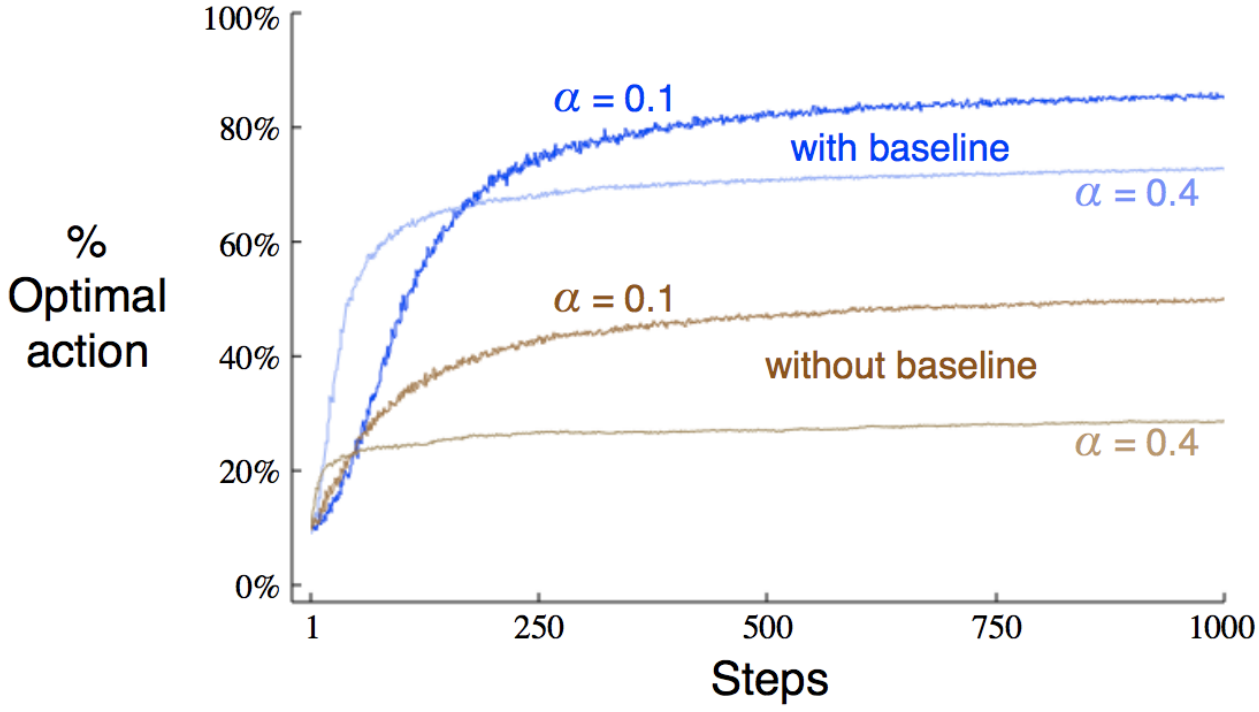
$$H_{t+1}(a) = H_t(a) + a(R_t - \overline{R}_t)(\mathbb{1}_{a=A_t} - \pi_t(a)), \quad \text{for all } a$$

source: Sutton & Barto, Reinforcement Learning: An Introduction.

- To conclude, use the quotient rule on derivatives to show the relation $\frac{\partial \pi_t(b)}{\partial H_t(a)} = \pi_t(b)$

$$
\begin{aligned}
\frac{\partial \pi_t(b)}{\partial H_t(a)} &= \frac{\partial}{\partial H_t(a)} \left[ \frac{e^{H_t(b)}}{\sum_{c=1}^{k} e^{H_t(c)}} \right] \\
&= \frac{\frac{\partial e^{H_t(b)}}{\partial H_t(a)} \sum_{c=1}^{k} e^{H_t(c)} - e^{H_t(b)} \frac{\partial \sum_{c=1}^{k} e^{H_t(c)}}{\partial H_t(a)}}{\left( \sum_{c=1}^{k} e^{H_t(c)} \right)^2} \\
&= \frac{\mathbb{I}_{a=b} e^{H_t(b)} \sum_{c=1}^{k} e^{H_t(c)} - e^{H_t(b)} e^{h_t(a)}}{\left( \sum_{c=1}^{k} e^{H_t(c)} \right)^2} \\
&= \frac{\mathbb{I}_{a=b} e^{H_t(b)}}{\sum_{c=1}^{k} e^{H_t(c)}} - \frac{e^{H_t(b)} e^{H_t(a)}}{\left( \sum_{c=1}^{k} e^{H_t(c)} \right)^2} \\
&= \mathbb{I}_{a=b} \pi_t(b) - \pi - t(b) \pi_t(a) \\
&= \pi_t(b)(\mathbb{I}_{a=b} - \pi_t(a))
\end{aligned}
$$

source: Sutton & Barto, Reinforcement Learning: An Introduction.




source: Sutton & Barto, Reinforcement Learning: An Introduction.

# Upper Confidence Bound Action Selection

- A downside of the $\varepsilon$ greedy action selection approach is that when exploring, it does not discriminate between the actions.
- One possible improvement consists in selecting among the actions during the exploration step, according to their potential for being optimal (and not completely randomly)
- One approach is to rely on the following upper confidence bound (UCB)

$$A_t = \underset{a}{\operatorname{argmax}} \left[ v_t(a) + c\sqrt{\frac{\ln(t)}{N_t(a)}} \right]$$

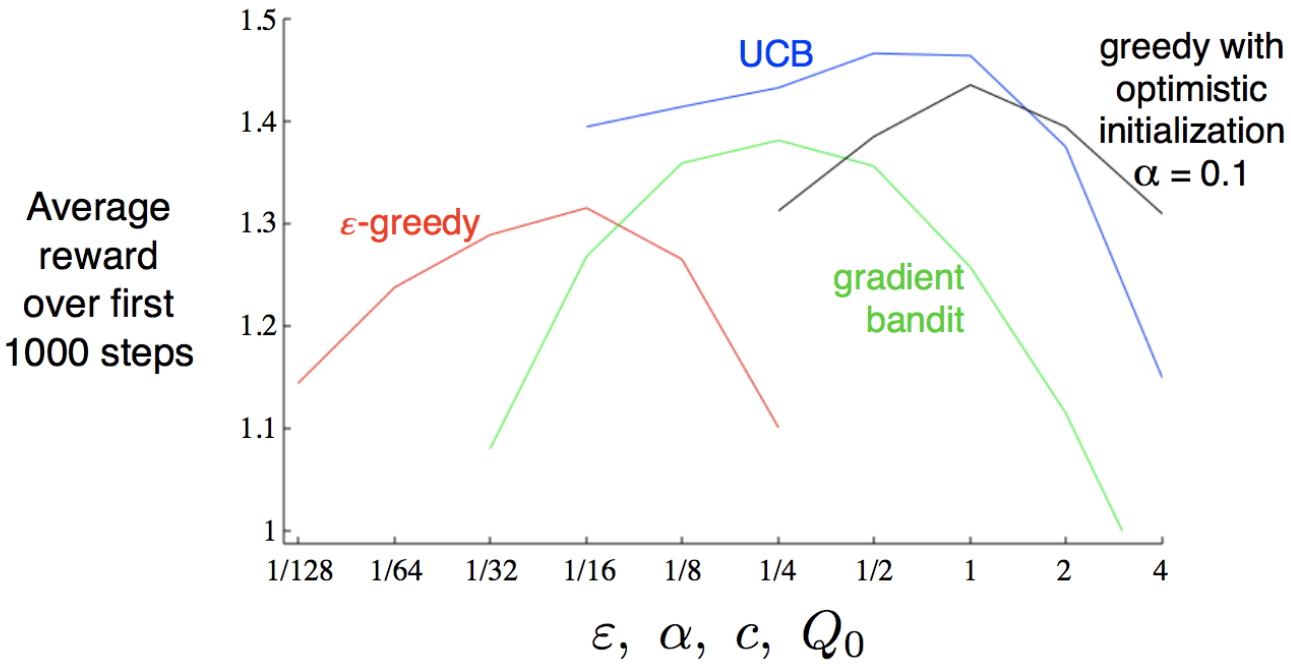
source: Sutton & Barto, Reinforcement Learning: An Introduction.

# Upper Confidence Bound Action Selection

- The idea of the UCB selection approach is to add a term (the $\sqrt{\log(t)/N_t}$ term) that accounts for the uncertainty in the value of an action $a$
- The objective that is maximized can be viewed as an upper bound on the potential value of the action $a$
- When the action $a$ is visited, the number $N_t(a)$ increases. On the opposite, when $a$ is not selected, the numerator $t$ increases while the number $N_t(a)$ remain constant, thus making this action more likely to get selected in future steps.
- After an infinite number of iterations, as the function is unbounded, every action will be selected at least once.

source: Sutton & Barto, Reinforcement Learning: An Introduction.

# Reinforcement learning: Comparison of Bandit algorithms




source: Sutton & Barto, Reinforcement Learning: An Introduction.

# Reinforcement learning

- So far in this chapter we have considered only nonassociative tasks, in which there is no need to associate different actions with different situation
- . In these tasks the learner either tries to find a single best action when the task is stationary, or tries to track the best action as it changes over time when the task is nonstationary
- The Multi-armed bandit problem is an instance of non associative learning.
- However, in a general reinforcement learning task there is more than one situation, and the goal is to learn a policy: a mapping from situations to the actions that are best in those situation

source: Barto, Sutton, and Brouwer, Biological Cybernetics, 1981.

# Associative vs Non Associative

- As an example, suppose there are several different n-armed bandit tasks, and that on each play you confront one of these chosen at random.
- The bandit task thus changes randomly from play to play.
- This would appear to you as a single, nonstationary n-armed bandit task whose true action values change randomly from play to play.
- And you could try using the simple Bandit algorithm (possibly non stationnary), but unless the true action values change slowly, these methods will not work very well.

source: Barto, Sutton, and Brouwer, Biological Cybernetics, 1981.

# Associative vs Non Associative

- Now suppose instead that when a bandit task is selected for you, you are given some information about its identity
- For example, let us assume you are facing a slot machine that changes the color of its display as it changes its action values.
- Now you can learn a policy associating each task, signaled by the color you see, with the best action to take when facing that task
- For instance, if red, play arm 1; if green, play arm 2.
- With the right policy you can usually do much better than you could in the absence of any information distinguishing one bandit task from another.

source: Barto, Sutton, and Brouwer, Biological Cybernetics, 1981.

# Associative vs Non Associative

- This is an example of an associative search task, so called because it involves both trial-and-error learning in the form of search for the best actions and association of these actions with the situations in which they are best

source: Barto, Sutton, and Brouwer, Biological Cybernetics, 1981.

# Associative vs Non Associative

- The approach that we followed so far for the Bandit algorithm gives a form of direct utility estimation (due to Widrow and Hoff and first introduced in adaptive control theory on the 1960's)
- The idea is that the utility of state is the expected total reward from that state onward. Each trial provides a sample of this quantity for each state visited
- Such a direct utility estimation succeeds in reducing the RL problem to an inductive learning problem but it misses a very important source of information
- The utilities of states are not independent: The utility of each state equals its own reward plus the expected utility of its successors

source: Barto, Sutton, and Brouwer, Biological Cybernetics, 1981.

# Associative vs Non Associative

- The utillity values obey the Bellman equation for a fixed policy

$$U^{\pi}(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi(s)) U^{\pi}(s')$$

- When the agent has several possible actions, one can extend this idea by learning an action utility function $Q[s, a]$. One can then again write a constraint equation that must hold at equilibrium when the $Q$ values are correct

$$Q[s, a] = R[s] + \gamma \sum_{s'} P(s'|s, a) \max_{a'} Q[s', a']$$

source: Barto, Sutton, and Brouwer, Biological Cybernetics, 1981.

# Associative vs Non Associative

- If we always choose to act according to the optimal action $a'$, we get the update rule

$$Q[s,a] \leftarrow Q[s,a] + \alpha(R[s] + \gamma \max_{a'} Q[s',a'] - Q[s,a])$$

  What this update rule is doing is correcting the utility so that it matches the Bellman equation

source: Barto, Sutton, and Brouwer, Biological Cybernetics, 1981.