Learning Algorithm Illustrations: From Simple to Deep

Leigh Tesfatsion

Professor Emerita of Economics
Courtesy Research Professor of Electrical & Computer Engineering
Iowa State University, Ames, IA 50011-1054
https://www2.econ.iastate.edu/tesfatsi/
tesfatsi@iastate.edu

Last Revised: 16 February 2024

References & Acknowledgement

Main References:

[1] **"Notes on Learning"**

https://www2.econ.iastate.edu/classes/econ308/tesfatsion/learning.Econ308.pdf

[2] **"Learning and the Embodied Mind"**

https://www2.econ.iastate.edu/tesfatsi/aemind.htm

Important Acknowledgement:

Some of the following slides are adapted from the following great online slide presentations:

Andrew Barto, *"Searching in the Right Space"*

Bill Smart, *"Reinforcement Learning: A User's Guide"*

Bill Tomlinson, *"Biomorphic Computing"*

Wendy Williams, *"GA Tutorial"*

Nicolas Galoppo von Borries, *"Intro To ANNs"*

Presentation Outline

❑ Overview

❑ Reactive Reinforcement Learning (RRL)

RRL Example 1: Deterministic RRL (e.g., Derivative-Follower)

RRL Example 2: Stochastic RRL (e.g., Roth-Erev algorithm)

❑ Belief-Based Learning (BBL)

BBL Example 1: Fictitious play learning

BBL Example 2: Hybrid forms (e.g., Camerer/Ho algorithm)

Presentation Outline...Continued

- ❑ Anticipatory Learning

  Example: Q-Learning

- ❑ Evolutionary Learning

  Example: Genetic Algorithms (GAs)

- ❑ Connectionist Learning

  Example: Artificial Neural Nets (ANNs)

Overview

- So far in Econ 308 we have worked with strategies for very simple one-stage and iterated (multi-stage) games

- The strategies we have seen to date for iterated games have been ***adaptive*** in the following sense:
  - ➔ The action dictated by the strategy at any given time is conditioned on the current (information) state of the player.

- However, this adaptation has been determined by a fixed rule in advance of any actual game play.

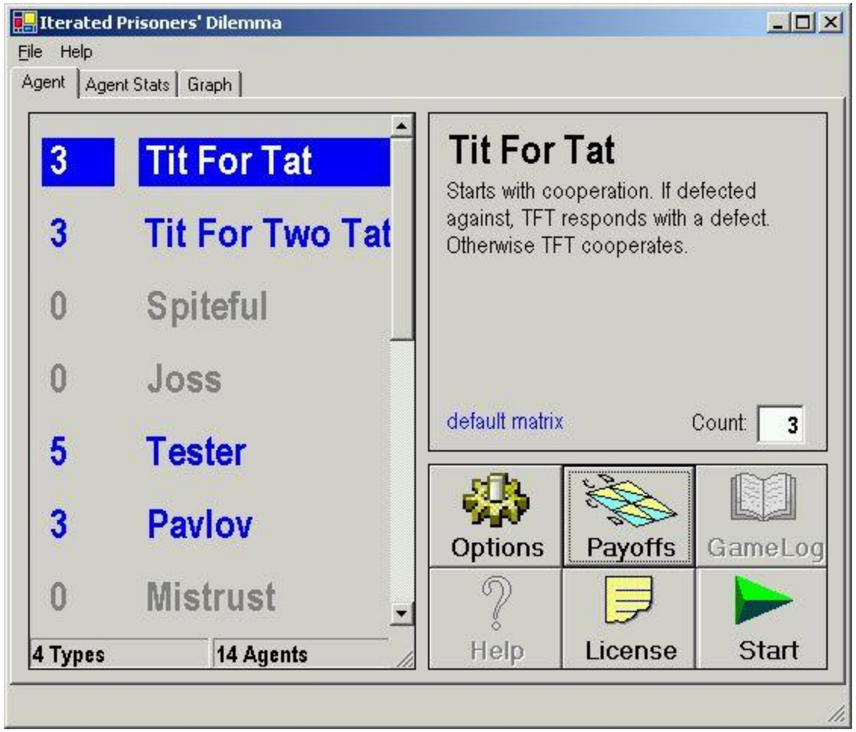
Example: Fixed rule defining the Tit-For-Tat (TFT) strategy

Play `cooperate' in the first stage. Then, in each successive stage, play the same move (`cooperate' or `defect') that your rival played in the previous stage.

Axelrod Tournament Demo

Basic Tournament by R. Axelrod; Demo developed by C. Cook

https://www2.econ.iastate.edu/tesfatsi/acedemos.htm



3	Tit For Tat
3	Tit For Two Tat
0	Spiteful
0	Joss
5	Tester
3	Pavlov
0	Mistrust

4 Types 14 Agents

Tit For Tat

Starts with cooperation. If defected against, TFT responds with a defect. Otherwise TFT cooperates.

default matrix

Count: 3

- User-specified strategies for playing a specified type of game (e.g., PD, Chicken, Stag Hunt) are pitted against one another in repeated round-robin play.
- **KEY ISSUE**

  What types of strategies **perform best over time**?

  Will **nasty or cooperative** types prevail?

Overview … Continued

- In the next part of Econ 308, we will investigate adaptive strategies for more complicated types of iterated market games.

- We will also investigate the possibility of learning in iterated market games.

- That is, we will want to permit one or more players to structurally modify their strategies (rules for play) during successive game iterations based on sequentially observed events.

Overview … Continued

Learning means …. for example:

- A player starts an iterated game with an initial strategy (“policy”) π dictating an action a to be taken in each state s:

  State s → Action a

- But, after observing the payoff (“reward”) r from using this state-action association, the player eventually decides to *change* this association:

  State s → Action a*

Caution: Intrinsic Ambiguity in the Distinction between Adaptation and Learning

- Suppose an agent is acting in accordance with a particular state-action association $s \rightarrow a$ in a general environment e.

- Suppose something happens (e changes to e*) that convinces the agent to change this association to some other association $s \rightarrow a^*$.

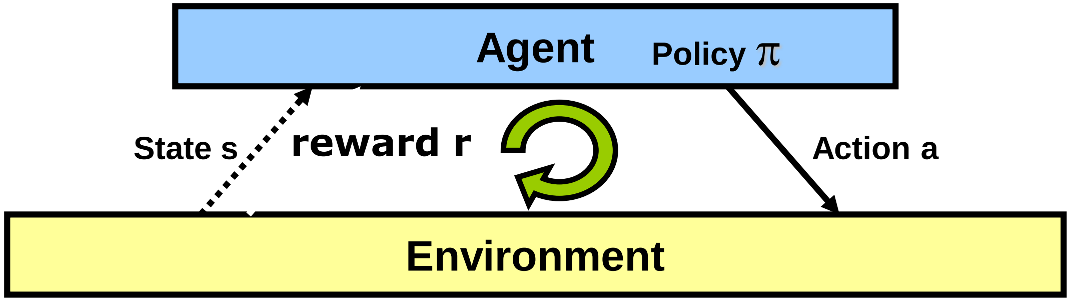
- If the definition of "state" is expanded from s to (s,e), the associations $(s,e) \rightarrow a$ and $(s,e^*) \rightarrow a^*$ ***have not changed***.

General Types of Learning

- ***Unsupervised Learning***
  - Update structure based on intrinsic motivation (curiosity, enjoyment, moral duty, …)
- ***Reinforcement Learning (RL)***
  - Update structure in response to successive rewards attained through actions taken
- ***Supervised Learning***
  - Update structure on basis of examples of desired (or required) state-action associations provided by an expert external supervisor

Reinforcement Learning (RL)

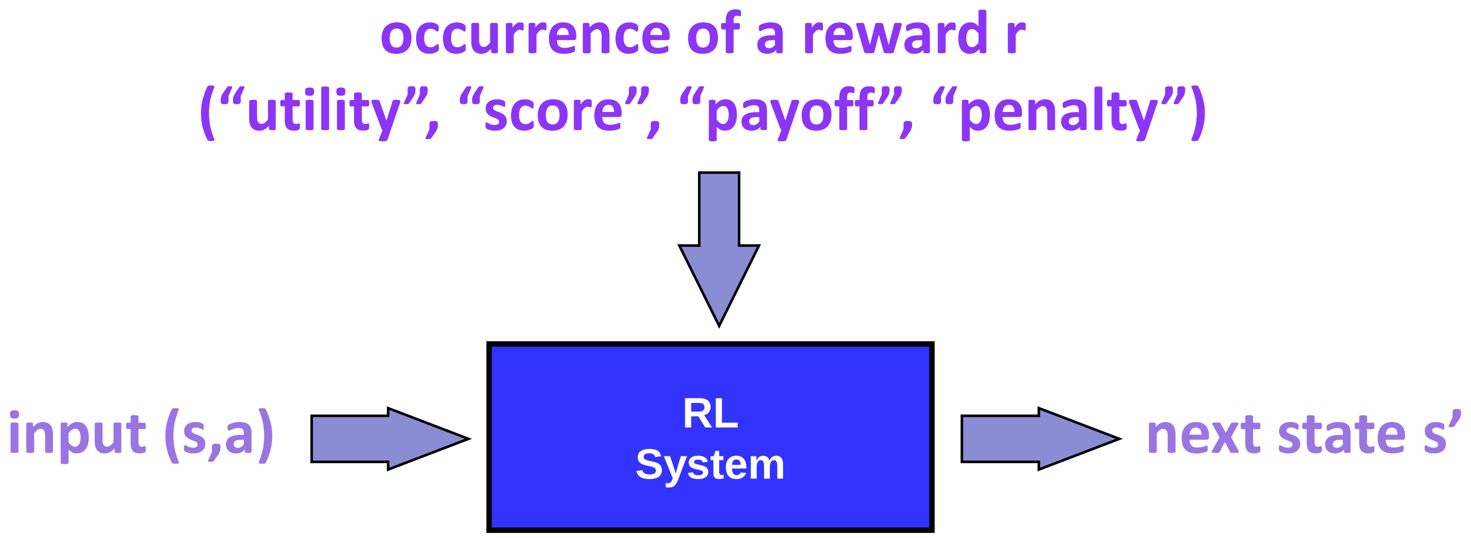
❑ Elements of traditional RL:



$$s_0 \xrightarrow{a_0 : r_0} s_1 \xrightarrow{a_1 : r_1} s_2 \xrightarrow{a_2 : r_2} \cdots$$

- **Policy π :** Maps each state s to an action choice a
- **Reward *r* :** Immediate value of state-action pairing
- **Transition model *T(s,a) = s'* :** Maps current state-action pairing (s,a) to a next state s'

Elements of Traditional RL …



occurrence of a reward r
(“utility”, “score”, “payoff”, “penalty”)

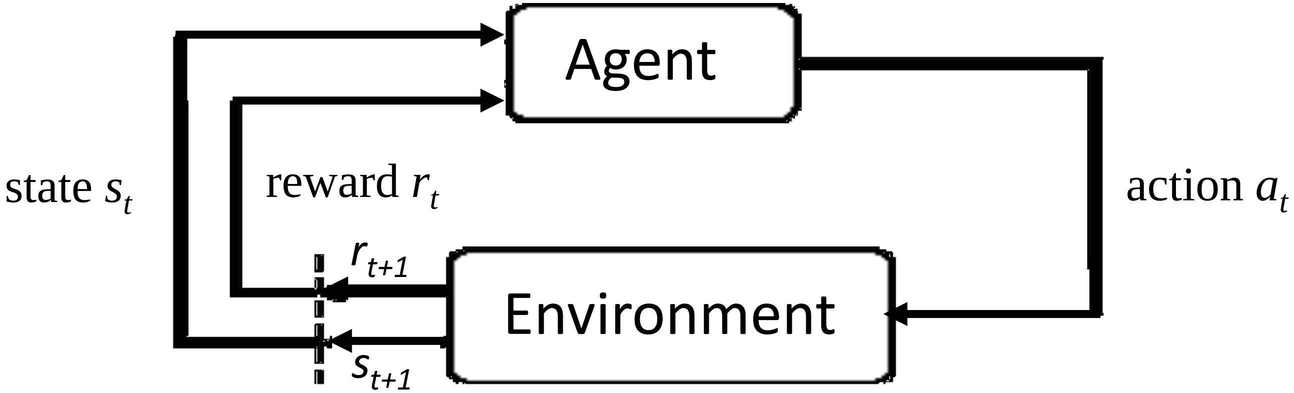
input (s,a)

RL
System

next state s’

Basic RL Intuition: The tendency to take an action *a* in a state *s* should be strengthened (reinforced) if it produces favorable results and weakened if it produces unfavorable results.

Traditional RL in More Detail



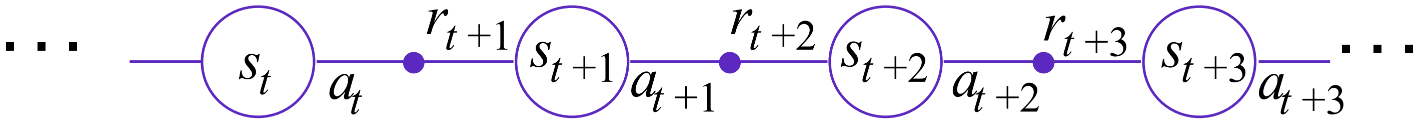
Agent and environment interact at discrete time steps: $t = 0, 1, 2, \ldots$

Agent observes state at step t: $s_t \in S$

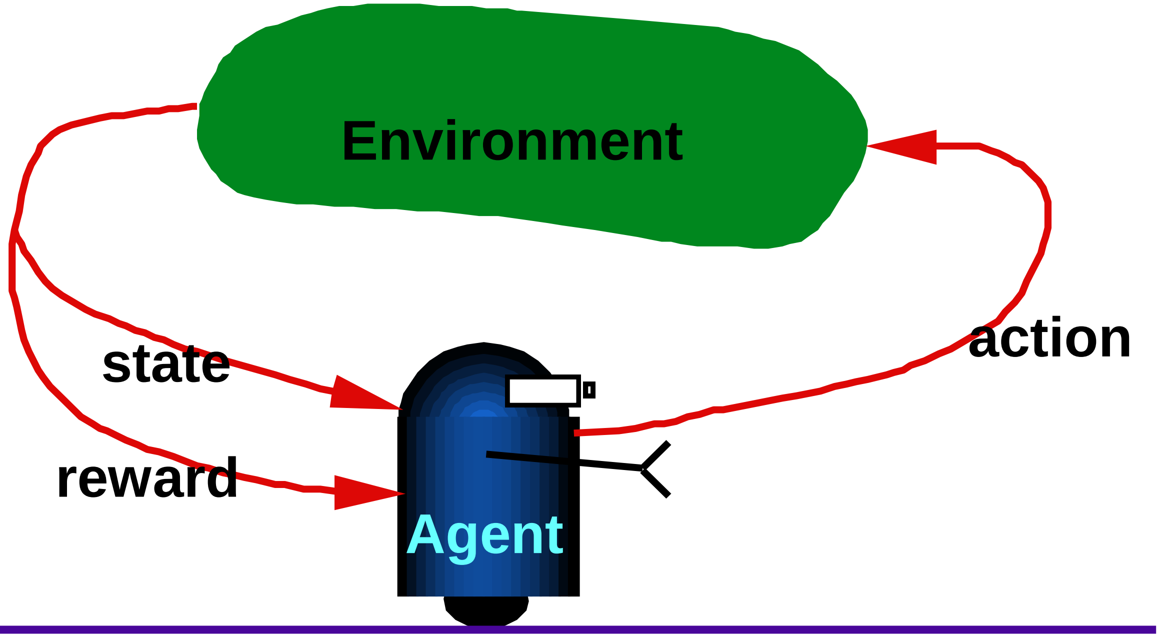
produces action at step t: $a_t \in A(s_t)$

gets resulting reward: $r_{t+1} \in \Re$

and resulting next state: s_{t+1}



Traditional RL View of Agent Action Choice



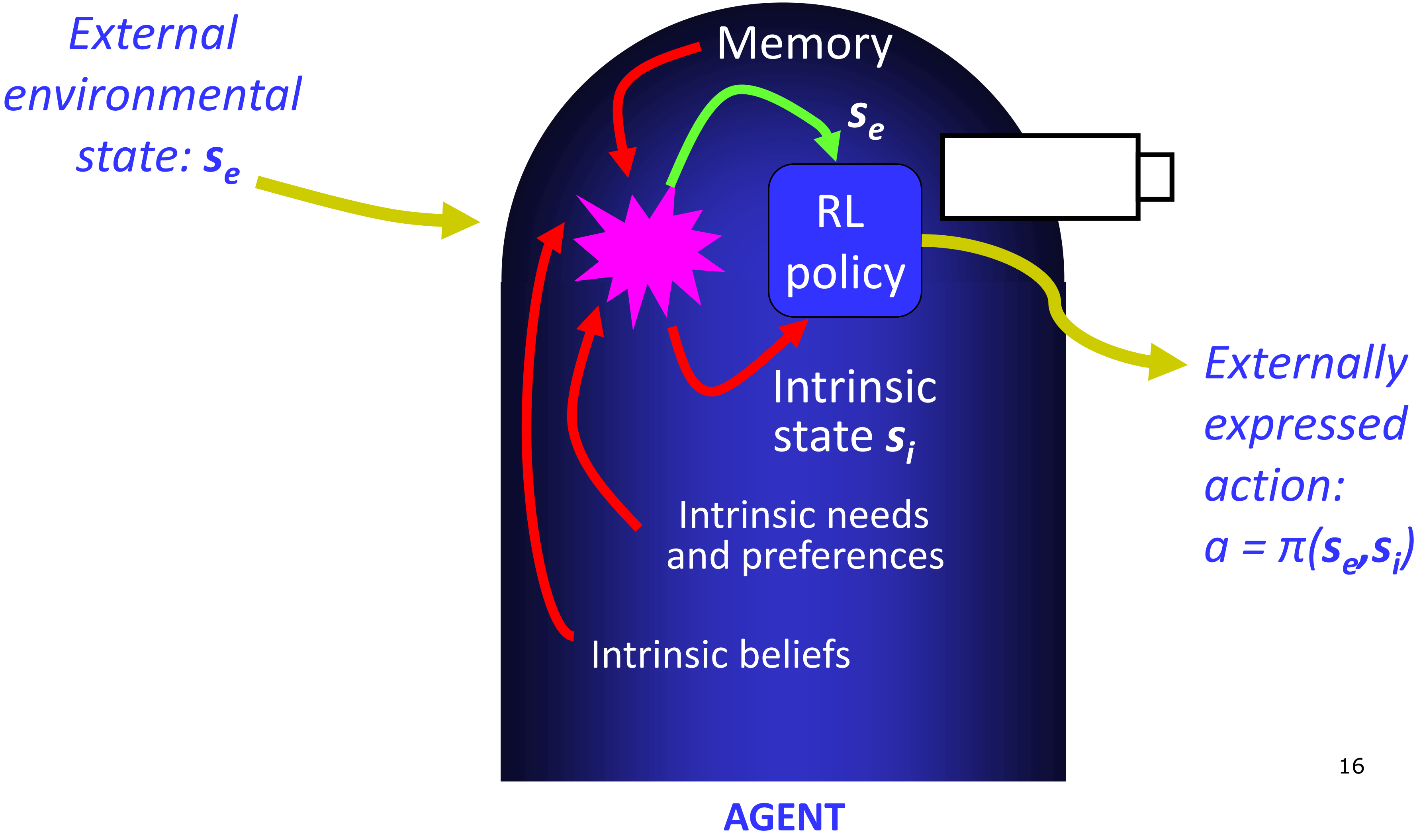
States and rewards are modeled as external forces determining an agent’s choice of actions.

In Accord with Human Motivation?

Factors that energize a person to act, and that direct his or her activity:

- *Extrinsic Motivation:* Being moved to act in hopes of receiving some external reward ($$, prize, praise, etc.)
- *Intrinsic Motivation:* Being moved to act because it is perceived to be inherently desirable, enjoyable, moral, …

More Modern Extrinsic/Intrinsic View of Agent Action Choice



External environmental state: s_e
Memory
s_e
RL policy
Intrinsic state s_i
Intrinsic needs and preferences
Intrinsic beliefs
Externally expressed action: $a = \pi(s_e, s_i)$
AGENT

Intrinsic Motivation: Questions

- An activity is intrinsically motivated if an agent does it for its own sake rather than to receive specific rewards (or avoid specific penalties)
- Curiosity, exploration, moral duty, . . .
- Can a *computational learning system* be intrinsically motivated?
- Specifically, can a *computational RL agent* be intrinsically motivated?

2. Reactive RL

Asks…

Given past events, what action should I take now?

Example 1: Deterministic Reactive RL

Derivative-Follower (DF) Adaptation

(Greenwald and Kephart, 1999)

- Originally developed as a simple way for computational agents to repeatedly modify a ***scalar*** decision *d*.
- The ***D***erivative-***F***ollower (***DF***) agent experiments with incremental increases or decreases in *d* of a given magnitude *Δd* > 0.
- An external reward *r* is attained after each change in *d*.
- The DF agent continues to move *d* in the same direction (increases or decreases) until the external reward *r* starts to decline, at which point the DF agent reverses the direction of movement in *d.*
- Letting states *s* be given by *Δr* and actions *a* be given by ± *Δd*, the associations $s \rightarrow a$ are in fact fixed in advance.

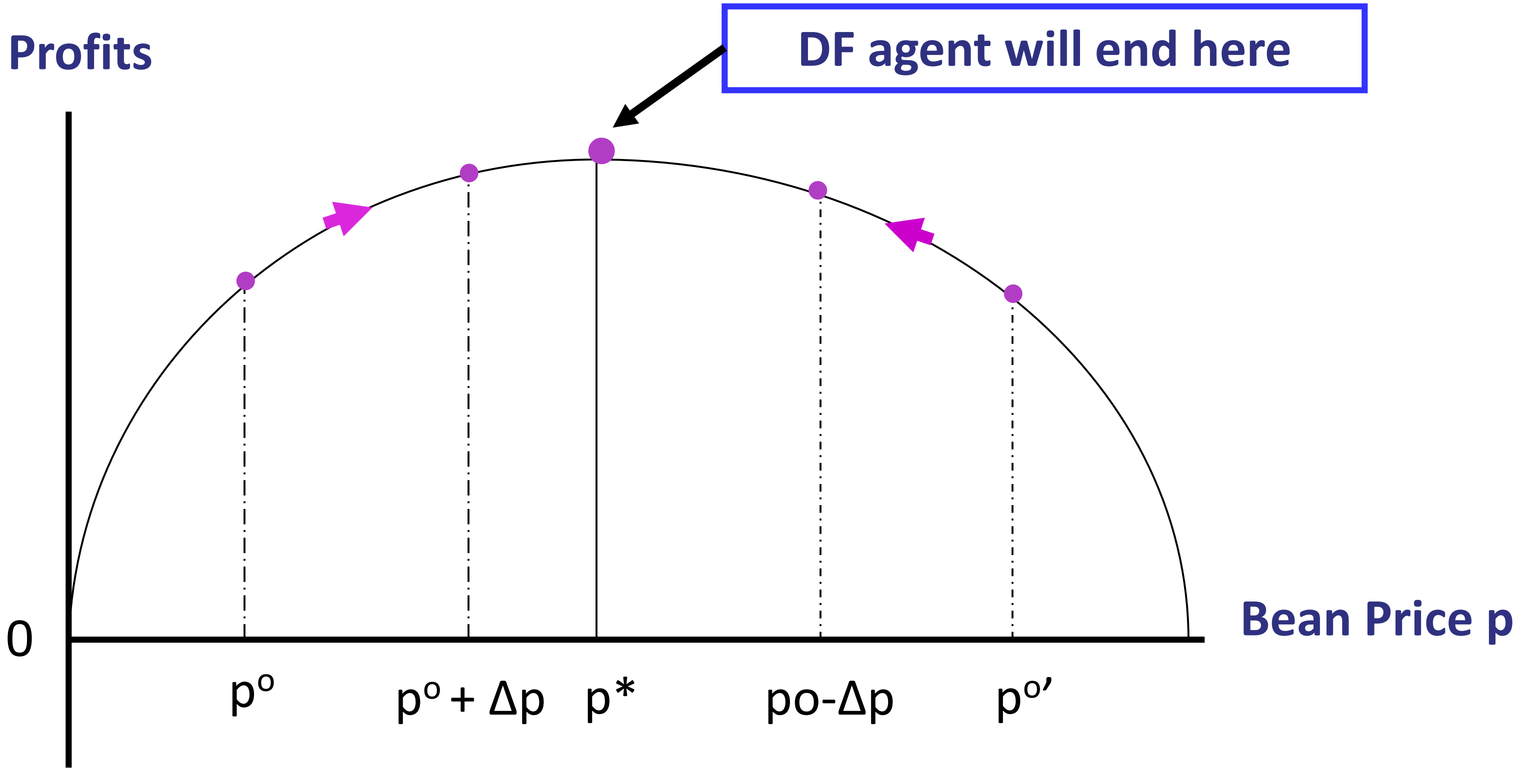
DF Adaptation: A Simple Market Example

- Each day a firm produces b* pounds (lbs) of beans.
- On the first day the firm selects an initial per-unit price p^o (U.S. dollars $ per pound) at which to sell b*.
- The firm then posts successively higher daily prices p for beans of the form $p^o+\Delta p$, $p^o+2\Delta p$, … , with $\Delta p > 0$, until resulting profits are observed to fall
- The firm then reverses course and starts to decrease p by step-size Δp. And so on…

❑ ***Question:*** Under what conditions will this DF adaptation learning process work well, if ever ?

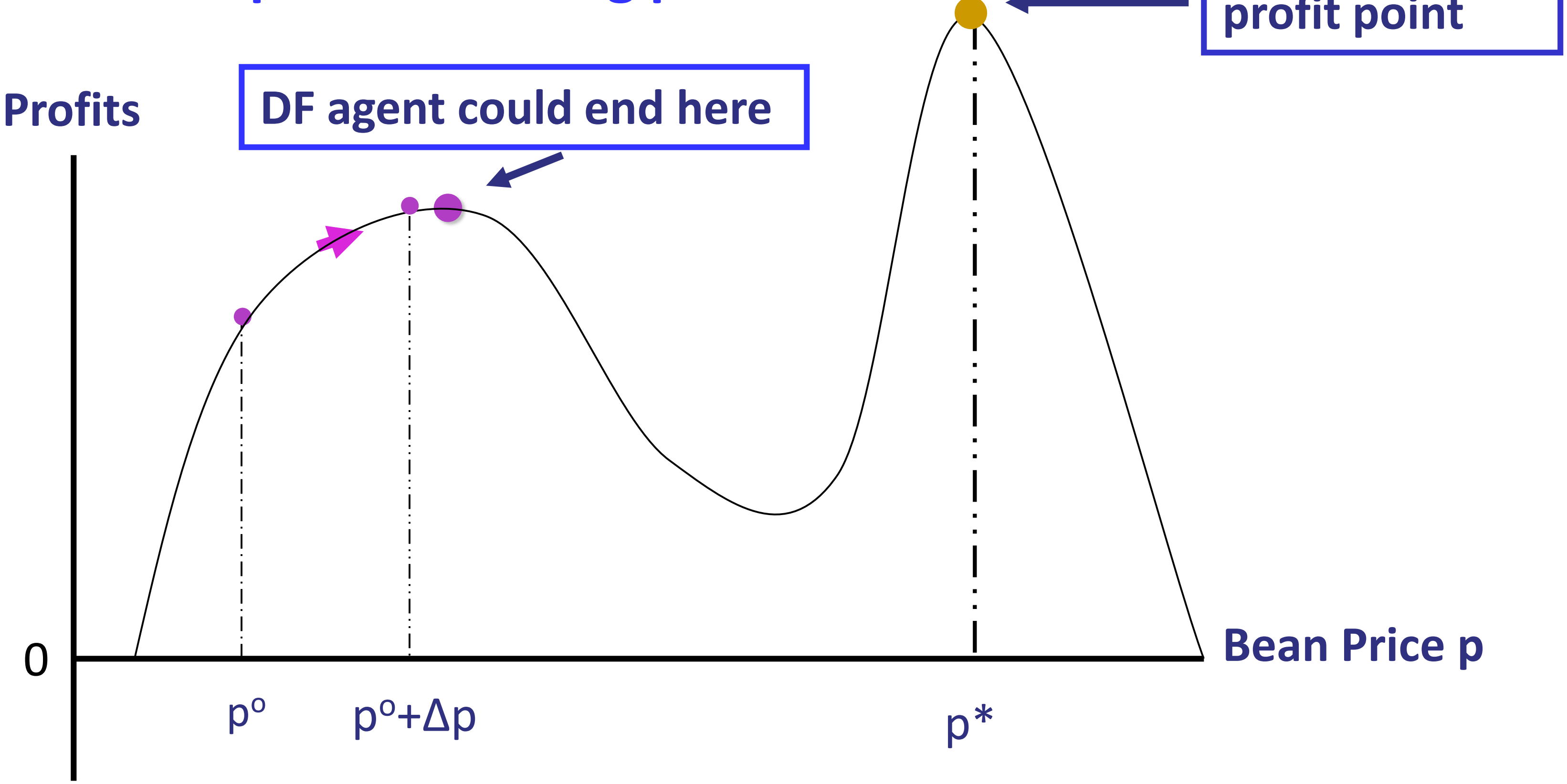
When will DF adaptation work well (if ever)?

- Suppose profits are a concave function of the price p



But suppose profits are *NOT* a concave function of the price p?

- □ Could end up on the wrong peak!

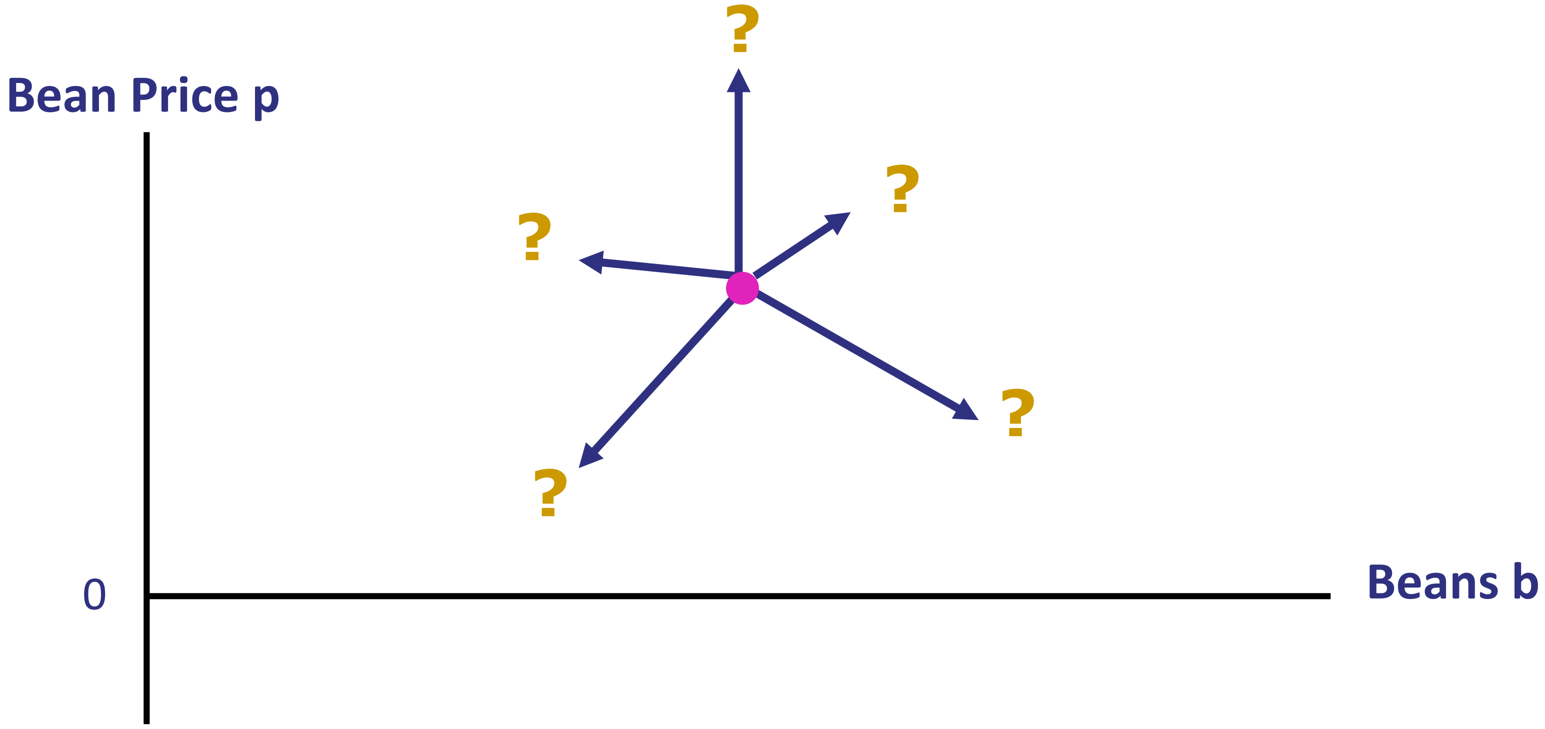


DF agent could end here

True maximum profit point

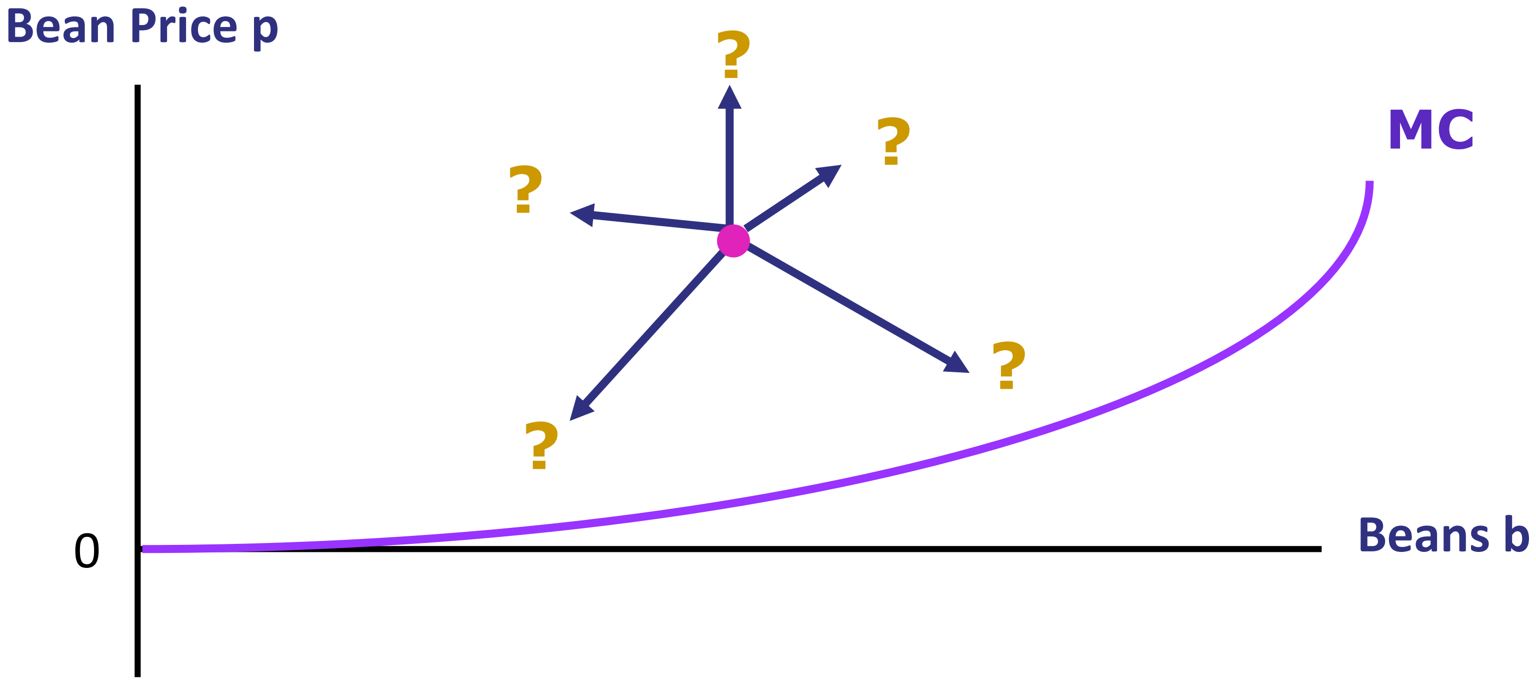
Or suppose a profit-seeking firm must set *BOTH* price *AND* quantity levels?

- Where to start, which direction to search in, and how far to search in this direction?



A profit-seeking firm should try to stay *on or above* its marginal production cost function MC

- ***KEY ISSUE:*** ***Correlated*** Δp and Δb choices needed to stay above MC and move in desirable directions



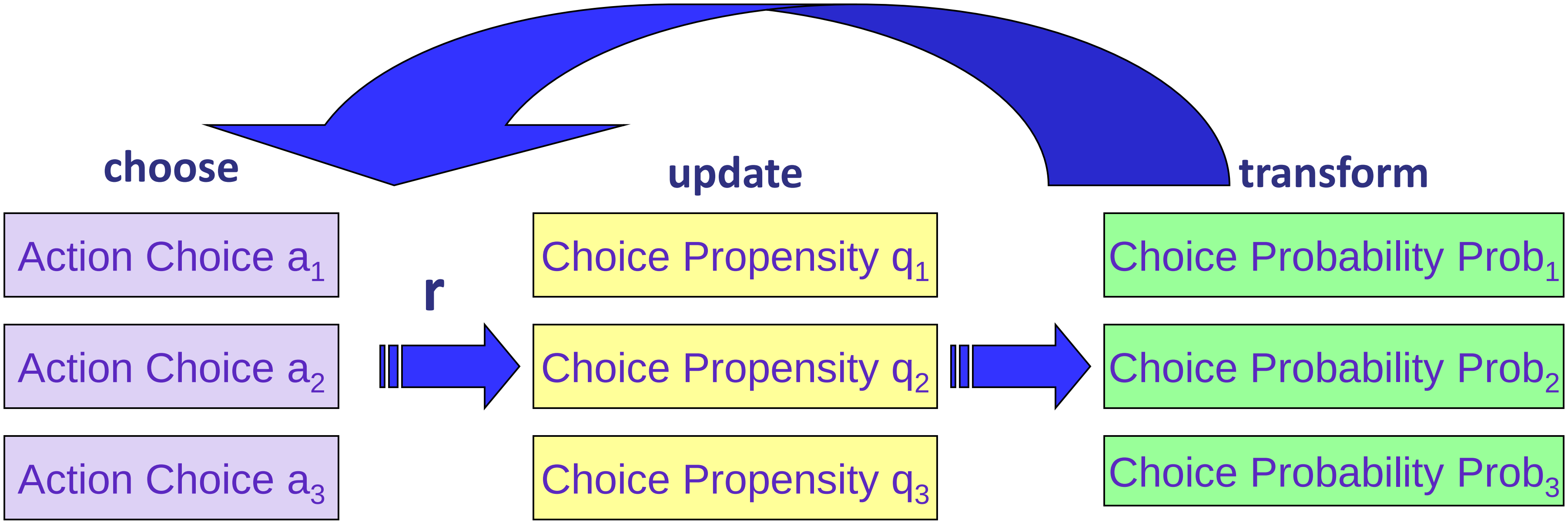
Example 2: Stochastic Reactive Reinforcement Learning based on Experimental Game Data

- Alvin E. Roth and Ido Erev (*Games & Economic Behavior*, 1995; *American Economic Review,* 1998)
  - Based on observations of people's behavior in iterated game play with ***multiple strategically interacting players*** in various game contexts
  - Roth/Erev determined that two extensions were needed in the RL methods developed earlier by psychologists for single decision makers learning in fixed environments:
    - Need to "forget" rewards received in distant past
    - Need for "spillover" of reward attributions across actions in early game play to encourage experimentation with new actions, thus avoiding premature fixation on a suboptimal action.

Roth-Erev Reinforcement Learning (RL): Basic Algorithm Steps

1. Initialize propensities q for choosing actions.
2. Generate action choice probabilities Prob from current action propensities q.
3. Choose an action a in accordance with current action choice probabilities Prob.
4. Update action propensity values q using the reward r received after the last chosen action a.
5. Repeat from step 2.

Roth-Erev RL: Depiction of Basic Algorithm Steps



choose

update

transform

Action Choice a_1

Action Choice a_2

Action Choice a_3

r

Choice Propensity q_1

Choice Propensity q_2

Choice Propensity q_3

Choice Probability Prob_1

Choice Probability Prob_2

Choice Probability Prob_3

- Action choice “a” leads to a reward “r”, followed by an updating of all action choice propensities “q” based on this reward, followed by a transformation of these action choice propensities into action choice probabilities “Prob”.

Roth-Erev RL: Updating of Action Choice Propensities

- Specification of initial propensity levels $q_j(1)$ for the possible action choices a_j of a decision-making agent in an initial time-step 1.
  - **Initial propensity levels act as "prior expected benefit" levels.**
  - High initial propensity levels → Agent will be disappointed with the unexpectedly low rewards resulting from his early action choices, which will encourage the agent to continue experimenting with other actions.
  - Low initial propensity levels → Agent will be happy with the unexpectedly high rewards resulting from his early action choices, which will encourage the agent to fixate prematurely on one of these actions.

Roth-Erev RL: Updating of trader choice propensities for possible action choices $\{a_1, \dots, a_N\}$ at each time-step t ≥ 1

Non-Negative Parameters:

- $q_j(1)$ Initial propensity for action a_j
- ϵ Experimentation parameter
- φ Recency parameter in [0,1]

Variables:

- t Current time-step
- $a_k(t)$ Actual action choice of trader at *t*
- $r_k(t)$ Trader's reward from action $a_k(t)$ at *t*
- $a_j(t)$ Possible action choice for trader at *t*
- $q_j(t)$ Trader's propensity for $a_j(t)$ at *t*
- N Number of possible action choices at *t*

Note: *Suppose reward $r_k(t)$ from action $a_k(t)$ chosen at t is zero. Then $E_j(\epsilon,N,k,t) = 0$ & $q_j(t+1)=[1-\varphi]q_j(t)$ for each possible action choice $a_j(t)$ in $\{a_1,\dots,a_N\}$. Thus, either all propensities $q_j(t)$ decrease ($0 < \varphi$) or all propensities $q_j(t)$ stay the same ($0 = \varphi$) from t to t+1.*

$$q_j(t+1) = [1-\phi]q_j(t) + E_j(\epsilon,N,k,t)$$

Response Function *E*

$$E_j(\epsilon,N,k,t) = \begin{cases} r_k(t)[1-\epsilon] & \text{if } j = k \\ r_k(t)\frac{\epsilon}{N-1} & \text{if } j \neq k \end{cases}$$

From Propensities to Probabilities

Example A: Probability *$Prob_j(t)$* of choosing action *$a_j(t)$* at time *t* =: Relative choice propensity for *$a_j(t)$* at time *t*

$$Prob_j(t) = \frac{q_j(t)}{\sum_{n=1}^{N} [\, q_n(t) \,]}$$

Implication of the Above Definition: *Let $A = \{a_1, \ldots, a_N\}$ denote the set of possible action choices at each time-step t. As seen on slide 29, if the reward $r_k(t)$ resulting from the action a_k chosen at a time-step t is zero,* then $E_j(\epsilon,N,k,t) = 0$ and *$q_j(t+1)=[1 - \varphi]q_j(t)$ for each action a_j in A. In this case, using above definition, $Prob_j(t) = Prob_j(t+1)$ for each action a_j in A since all the "[1- φ]" factors cancel out of $Prob_j(t+1)$.*

Example B: Gibbs-Boltzmann Probability

- Handles negative action choice propensity values $q_j(t)$
- *Let T =: Temperature ("cooling") parameter, T* > 0
- *T* affects dynamic shape of probability distributions

$$Prob_j(t) = \frac{e^{q_j(t)/T}}{\sum_{n=1}^{N} e^{q_n(t)/T}}$$

More on the Updating of Roth-Erev RL Action Propensities …

- In time-changing environments, decision makers might want to "forget" rewards r received in the distant past:
  - **Forgetting in Roth-Erev RL is controlled by a "recency" parameter φ that lies between 0 and 1**
  - As φ approaches 1, the ***heaviest*** weight is assigned to the ***most recently*** received rewards r
  - As φ approaches 0, approximately equal weight is assigned to each reward r that has been received to date
  - If φ = 0 and ϵ = 0, exactly equal weight is assigned to each reward r that has been received to date.

More on the Updating of Roth-Erev RL Action Propensities …

- Need "spillover" of reward attributions across actions in early game play to encourage experimentation and to avoid premature fixation on suboptimal chosen actions.
  - **Spillover in Roth-Erev RL is controlled by an "experimentation" parameter ϵ that lies between 0 and 1.**
  - As ϵ increases, there is more "spillover" of the reward resulting from a chosen action a_k to non-chosen actions a_j, resulting in smaller divergence among choice propensities q_k and q_j
  - As ϵ approaches 0, the reward resulting from a chosen action a_k is attributed only to a_k, hence only a_k's propensity q_k is updated.

Modification of the Roth-Erev RL Response Function *E*

- Nicolaisen, Petrov & Tesfatsion (*IEEE Transactions on Evolutionary Computation*, 2001) modified the Roth-Erev RL response function *E* -- as shown below -- to permit updating of action choice propensities to occur in response to each received reward *r*, even if *r = 0*. **(Compare with slide 29.)**
- Let A = {a_1, …, a_N} denote the set of possible trader action choices $a_j(t)$ at *t*, and let $a_k(t)$ = the actual trader action choice at *t*. Define a ***modified response function for t*** as follows:

$$EM_j(\epsilon, N, k, t) = \begin{cases} r_k(t)[1-\epsilon] & \text{if } j = k \\ q_j(t)\frac{\epsilon}{N-1} & \text{if } j \neq k \end{cases}$$

Suppose this modified response function is used in place of $E_j(\epsilon,N,K,t)$ defined on slide 29, and $\varphi \approx 0$. Then, if the action a_k chosen at time-step *t* results in a reward $r_k(t) = 0$, the action propensity $q_k(t+1)$ for a_k at time-step t+1 is (approximately) unchanged from time-step *t* whereas the action propensity $q_j(t+1)$ for each other possible action choice a_j tends to increase relative to time-step t. Thus, movement away from a_k is encouraged.

Modified Roth-Erev RL: Experimental Results

- In the following study, traders in test cases typically achieved high market efficiency (≥ 90%) using the ***Modified*** Roth-Erev RL algorithm and much lower market efficiency (e.g. 20%) using the ***Original*** Roth-Erev RL algorithm.

J. Nicolaisen, J., V. Petrov, and L. Tesfatsion, "Market Power and Efficiency in a Computational Electricity Market with Discriminatory Double-Auction Pricing," *IEEE Transactions on Evolutionary Computing,* Vol. 5 (October 2001), pp. 504–523.

- Similar comparative performance findings are reported in:

M. Pentapalli, "A Comparative Study of Roth-Erev and Modified Roth-Erev Reinforcement Learning Algorithms for Uniform-Price Double Auctions,"
M.S. Thesis, March 2008.

https://www2.econ.iastate.edu/tesfatsi/MridulPentapalli.MSThesisTalk2008.pdf

3. Belief-Based Learning (BBL)

Asks…

What ***different*** rewards might I have received in the past if I had acted differently?

And how can I use these ***"opportunity cost"*** assessments to help choose a better action now?

Belief-Based Learning …

- In belief-based learning, the presence of other decision-making agents in the learning environment is explicitly considered.
- Variants of belief-based learning currently in use by economists include:
  - **Cournot (naïve) belief learning** – the belief that rivals will act today in the same way they acted in the immediate past
  - **Fictitious play** – the belief that rivals will act today in accordance with the historical frequencies of all their past action choices.
  - **Experience-weighted attraction learning** (Camerer/Ho 1999) – hybrid of reactive RL and fictitious play learning

Belief-Based Learning: Example 1
Fictitious Play Learning (FPL)

- An agent A assumes each other agent in its choice environment chooses its actions in accordance with an unknown but time-invariant "probability distribution function (PDF)".

- Agent A estimates these PDFs based on the historical frequencies with which other agents have been observed to choose different actions.

- At each time t, Agent A chooses a "best response" action conditional on its current PDF estimates for other agents.

Fictitious-Play Learning (FPL):
An Illustrative Matching Pennies Game

		Player 2	
		Heads	Tails
Player 1	Heads	(1,-1)	(-1,1)
	Tails	(-1,1)	(1,-1)

FPL Illustration:
Matching Pennies … *Continued*

- The one-shot matching pennies game has NO Nash equilibrium in "pure strategies".

- That is, none of the four feasible action pairs (H,H), (H,T), (T,H), or (T,T) is a Nash equilibrium.

- However, suppose Player 1 is choosing its actions H and T in accordance with a ***mixed strategy***, i.e., a probability distribution defined over the action domain {H,T} that takes the form [Prob^1(H), Prob^1(T)].

- Then Player 2 can calculate a "best response" mixed strategy [Prob^2(H),Prob^2(T)] to Player 1's mixed strategy that maximizes Player 2's ***expected*** payoff.

FPL Illustration:
Matching Pennies … *Continued*

□ Player 2 is said to engage in ***Fictitious Play Learning (FPL)*** in the matching pennies game if the following conditions hold:

- The game is played in successive periods t=1,2,…, and Player 2 in each period t > 1 knows the actions that have been chosen by Player 1 in all **past** periods.
- In each period t > 1, Player 2 forms an estimate of the mixed strategy it thinks is being used by Player 1 based on the frequencies with which Player 1 has been observed to choose H and T in past game plays.
- In each period t > 1, Player 2 chooses a "best response" mixed strategy for its own action choice conditional on its current estimate for the mixed strategy being used by Player 1.

FPL Illustration:

Matching Pennies … *Continued*

- **EXAMPLE:** Suppose Player 1 has selected H and T with the following frequencies over the PAST ten periods t= 1,…,10
  - Action H: 5 times
  - Action T: 5 times

- Then Player 2's CURRENT (t=11) estimate for the mixed strategy being used by Player 1 to choose an action is
  - $\text{Prob}^1(H) = 5/10 = 1/2$
  - $\text{Prob}^1(T) = 5/10 = 1/2$

- Player 2's best response to this estimated mixed strategy for Player 1 is the mixed strategy $\text{Prob}^2(H) = 1/2$, $\text{Prob}^2(T) = 1/2$.

- **NOTE:** It can be shown that this pair of ***mixed*** strategies is ***the unique Nash equilibrium for the one-shot matching pennies game.***

Open Issues for Fictitious-Play Learning (FPL)

- Determination of estimated mixed strategies for other players is straightforward <u>if</u> all past action choices are observed.

- But how, practically, to calculate a “best response” mixed strategy in each time-period, given realistic time and cost constraints?

- And what happens if other players are <u>not</u> using time-invariant mixed strategies to choose their action choices?

Example 2: Experience-Weighted Attraction (EWA) Algorithm

(Camerer and Ho, *Econometrica*, 1999)

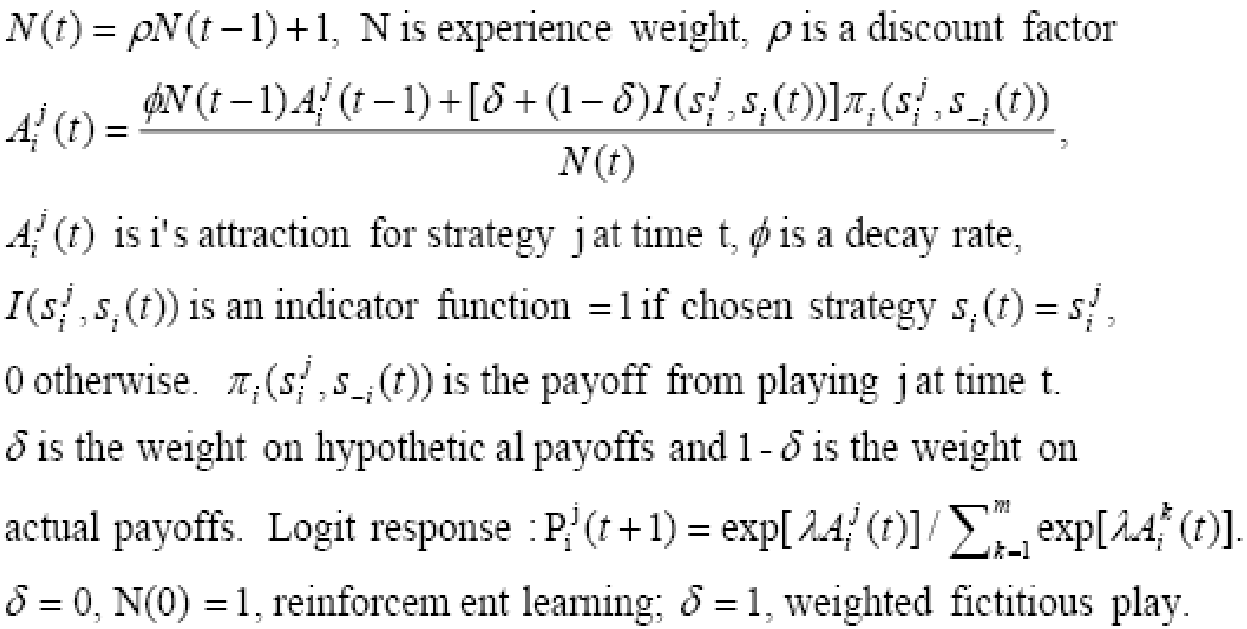
- ❑ Reactive Reinforcement Learning (RRL) assumes agents only consider <u>actual</u> past rewards, ignoring consideration of opportunity costs, i.e., ignoring consideration of rewards that <u>might</u> have been obtained had <u>different</u> actions been taken.

- ❑ Fictitious-Play Learning (FPL) assumes agents form opportunity cost estimates to select best-response mixed strategies.

- ❑ EWA is a <u>hybrid learning form</u> that combines RRL & FPL.

EWA Algorithm …

- The EWA Algorithm assumes propensities (“attractions”) and probabilities (“logit responses”) for (mixed) strategy choices are sequentially generated as follows:



$N(t) = \rho N(t-1) + 1,$ N is experience weight, ρ is a discount factor

$$A_i^j(t) = \frac{\phi N(t-1) A_i^j(t-1) + [\delta + (1-\delta) I(s_i^j, s_i(t))] \pi_i(s_i^j, s_{-i}(t))}{N(t)},$$

$A_i^j(t)$ is i's attraction for strategy j at time t, ϕ is a decay rate,

$I(s_i^j, s_i(t))$ is an indicator function $=1$ if chosen strategy $s_i(t) = s_i^j$,

0 otherwise. $\pi_i(s_i^j, s_{-i}(t))$ is the payoff from playing j at time t.

δ is the weight on hypothetic al payoffs and 1 - δ is the weight on

actual payoffs. Logit response : $\mathrm{P}_i^j(t+1) = \exp[\lambda A_i^j(t)] / \sum_{k=1}^{m} \exp[\lambda A_i^k(t)]$.

$\delta = 0$, N(0) = 1, reinforcem ent learning; $\delta = 1$, weighted fictitious play.

4. Anticipatory Learning

Asks…

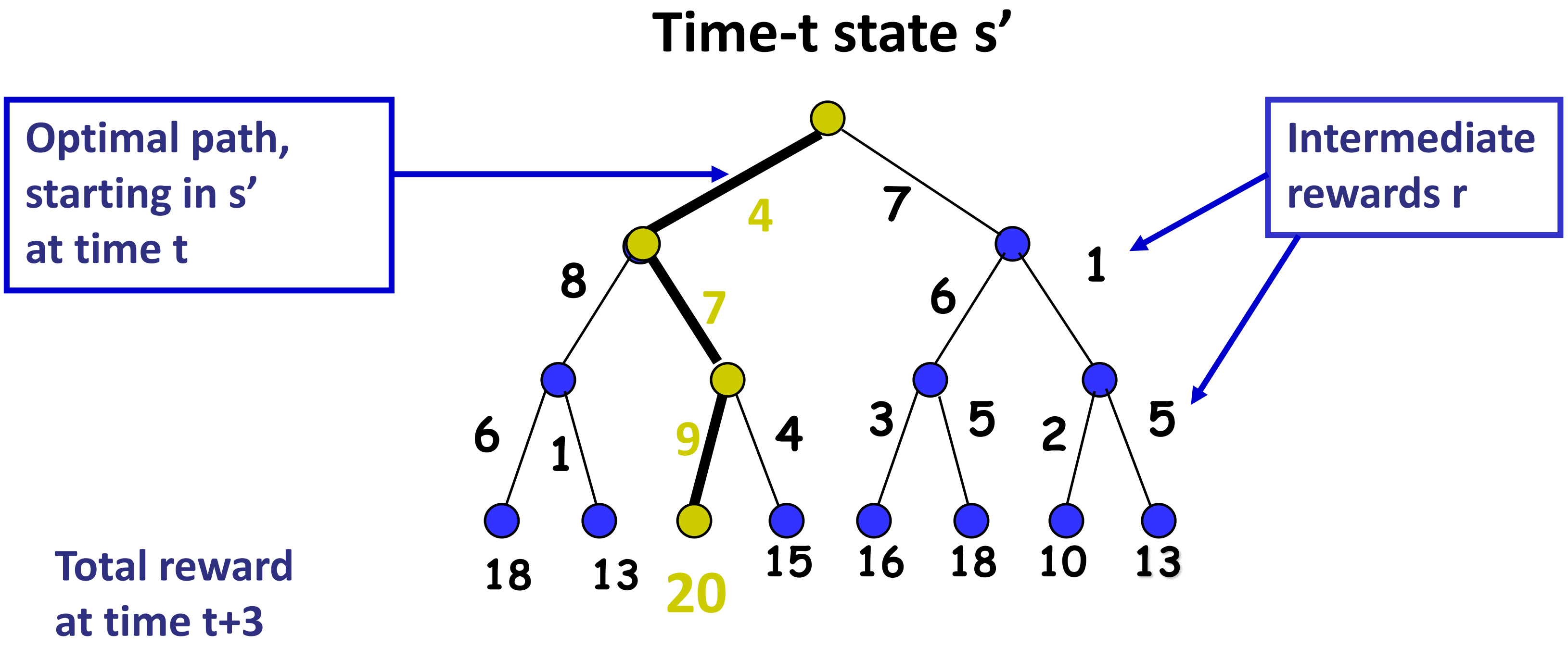
If I take this action now, what might happen in the future?

Key Anticipatory Learning Concept:
Value Function

Let the *optimum total future reward* obtainable by a decision-making agent, starting at time t in some state s’, be denoted by

$$V_t(s')$$

Value Function Illustration



Time-t state s’

Optimal path, starting in s’ at time t

Intermediate rewards r

Total reward at time t+3

Value function V_t gives $V_t(s') = 20$ if the decision tree ends at [t+3]

(*Total reward* = Sum of all intermediate rewards r)

Key Idea: Derive a Recursive Relationship Among Successive Value Functions

- ❑ Suppose I am currently in state s’ at time t.
- ❑ Suppose I take an action a’, get a reward r’ = R(s’,a’), and transit to a new state s’’ = T(s’,a’).
- ❑ Then, the best I can do starting from time t+1 is

$$V_{t+1}(\, s'')$$

- ❑ Consequently, the best I can do ***starting from time t*** is

$$V_t(s') = \max_a [\, R(s',a) + V_{t+1}(T(s',a))\,]$$

More Formally Stated:

Richard Bellman's Famous Principle of Optimality

(Dynamic Programming, 1950s)

- Let t denote the "current time" and let S = {s,s',...} denote the collection of all possible states of the world at time t.

- For each state s in S, let A(s) = {a,a',...} denote the collection of all feasible actions that an agent can take in state s at time t.

- For each state s in S, let W denote the collection of all possible total rewards w an agent can attain over current and future times t,...,TMax.

- Let the ***value function*** V_t:S→W be defined as follows: For each s in S, V_t(s) gives the optimum total reward w in W that can be attained by the agent over current and future times t,...,TMax starting in state s at time t.

Principle of Optimality…Continued

- Let π* denote the ***optimal policy function*** giving the optimal action a' as a function a'=π*(t,s') of the current time t and state s'.
- Let T denote the ***transition function*** that determines the next state s'' as a function s''=T(s',a') of the current state s' and the current action choice a'.
- Let R denote the ***intermediate return function*** that determines the immediate reward r'' as a function r''=R(s',a') of the current state s' and current action choice a''.
- Then for each state s' in S:

$$V_t(s') \;=\; R(s',\pi^*(t,s')) \;+\; V_{t+1}(\,T(s',\pi^*(t,s'))\,)$$

$$=\; \mathrm{Max}_{\mathbf{a}}\,[\,R(s',a) \;+\; V_{t+1}(T(s',a))\,]$$

Practical Difficulties

- ❑ How practically to compute the optimal policy function π^* ?
- ❑ What if the transition function T is not known? And what if state transitions depend on actions chosen by **MANY** agents, not just by me?
- ❑ What if the return function R is not known?
- ❑ How practically to compute the value function V?

One Possible Approach: Replace V-values by Q-values (Watkins, 1989)

- Suppose the final time TMax is infinite and suppose that π*, T, R, and V are independent of time t. ***Note:*** These are strong assumptions!

- For each s in S and each a in A(s), define

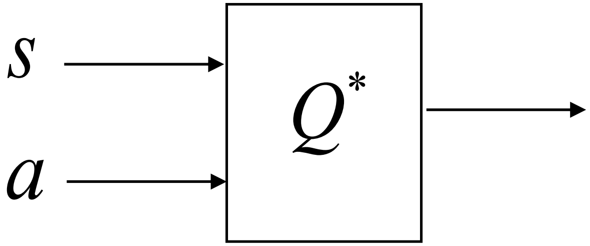
  $$Q^*(s,a) = [\,R(s,a) + V(T(s,a))\,]$$

- If these Q*-values can be learned, the optimal policy function π* can be found without knowing the T, R, and V functions, as follows: For any s' in S,

  π*(s') = action a' that maximizes Q*(s',a) over a in A(s')

- But will π* result in good action choices if state/reward outcomes in fact depend on actions of multiple agents?

Q-Learning in More Detail



Q values

$$Q^*(s,a)$$

expected return for taking action *a* in state *s* and then following an optimal policy thereafter

For any state *s*, any action *a** that maximizes *Q*(s,a)* is called an **optimal action**:

*a** = [optimal action in state s] $= \arg\max_a Q^*(s,a)$

Let $Q(s,a) =$ current estimate of $Q^*(s,a)$

Q-Learning …

Q-learning in its simplest form iteratively determines estimates Q(s,a) for Q*(s,a) conditional on a user-specified ***learning rate a* , 0 ≤ *a* ≤ 1** .

- Q-learning avoids direct calculation of T, R, V
- The Q-value estimates Q(s,a) are stored in a table
- The Q-value estimates are updated after each new observation is obtained.
- The Q-value estimates depend on observation history but not directly on the particular method used to generate action choices.

Basic Q-Learning Algorithm

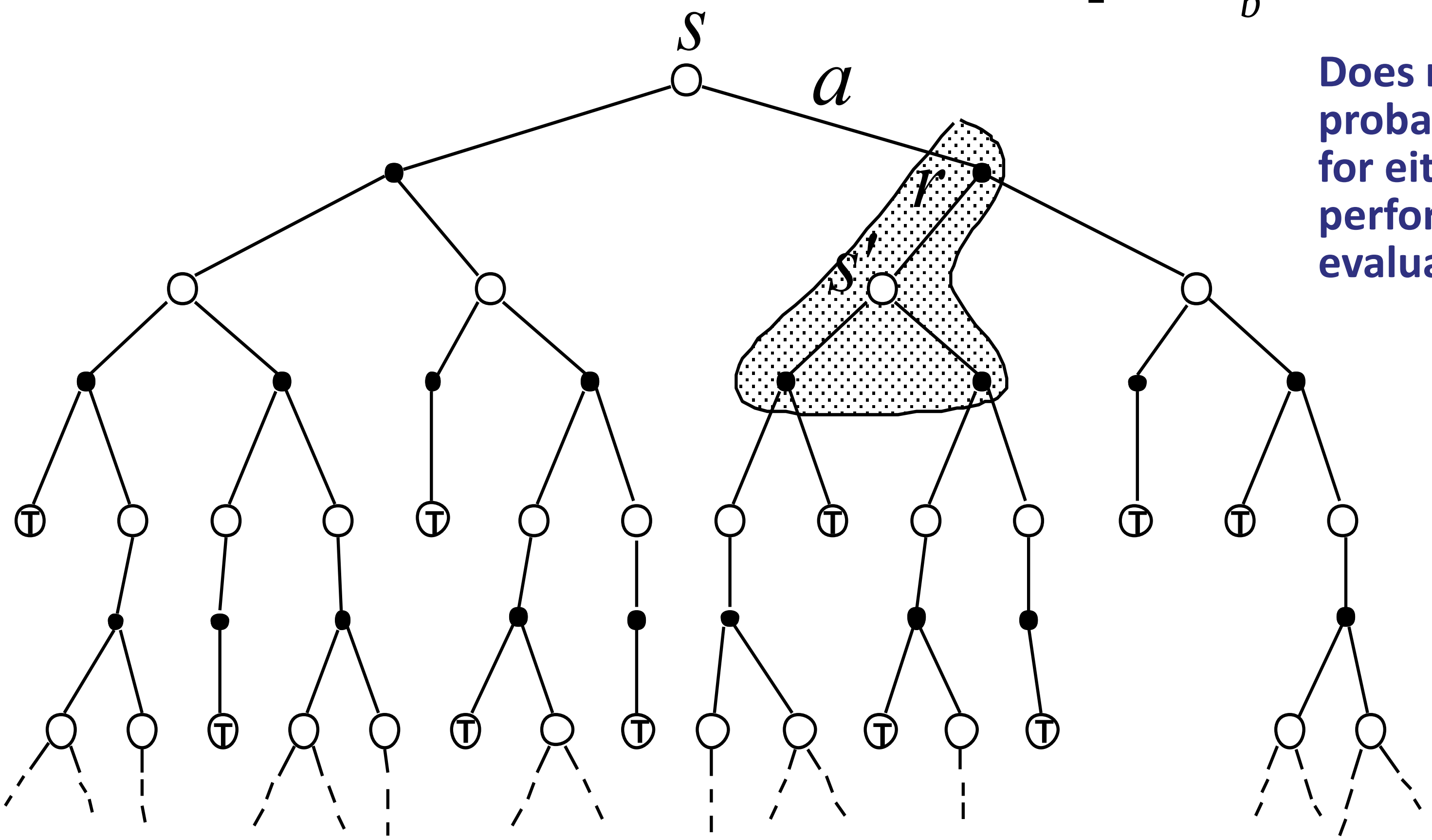
1. Initialize Q(s,a) to a random value for each state s in S and each action a in A(s).
2. Observe actual state s'.
3. Pick an action a' in A(s') and implement it.
4. Observe next state s'' and next reward r''.
5. Update Q(s',a') value as follows:

$$Q(s',a') \leftarrow [1 - a]Q(s',a') + a[\, r'' + \max_a Q(s'',a) \,]$$

6. Loop back to step 2.

Q-Learning Update Process

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha\left[r + \max_{b} Q(s',b)\right]$$



Does not need a probability model for either learning or performance evaluation

Picking Actions for Q-Learning

- Just as in reactive RL, an agent might want to pick “optimal” actions most of the time but also do some exploration.
  - An agent can exploit its ***current*** information state s to choose a “greedy” action a in A(s) that ***currently*** appears to be optimal.
  - But the agent might also choose an action for exploratory purposes, to learn more about its choice environment.
  - Exploring might permit the agent to learn a better policy π:s→ a(s) for determining ***future*** action choices.
  - This is called the ***exploration/exploitation problem***

Picking Actions for Q-Learning …

- ***e-Greedy Approach***
  - Given state s, choose an action a in A(s) with the highest value Q(s,a) with probability 1-e and explore (pick a random action) with probability e
- ***Gibbs-Boltzmann (soft-max) approach***
  - Given state s, pick action a in A(s) with probability

$$P(a|s) = \frac{e^{\left(\frac{Q(s,a)}{\tau}\right)}}{\sum_{a'} e^{\left(\frac{Q(s,a')}{\tau}\right)}}$$

where τ = "temperature"

5. Evolutionary Learning

Asks…

Given all the actions that have been taken to date by myself (and possibly by others), together with observations on the rewards that have resulted, what ***NEW*** actions might I devise to try to do better?

Evolutionary Learning Algorithms

EXAMPLES:

- Genetic Algorithm (GA) – John Holland 1970s
- Genetic Programming (GP) – John Koza 1990s
- Evolutionary Strategy (ES) – Rechenberg 1970s
- Evolutionary Program (EP) … Etc.

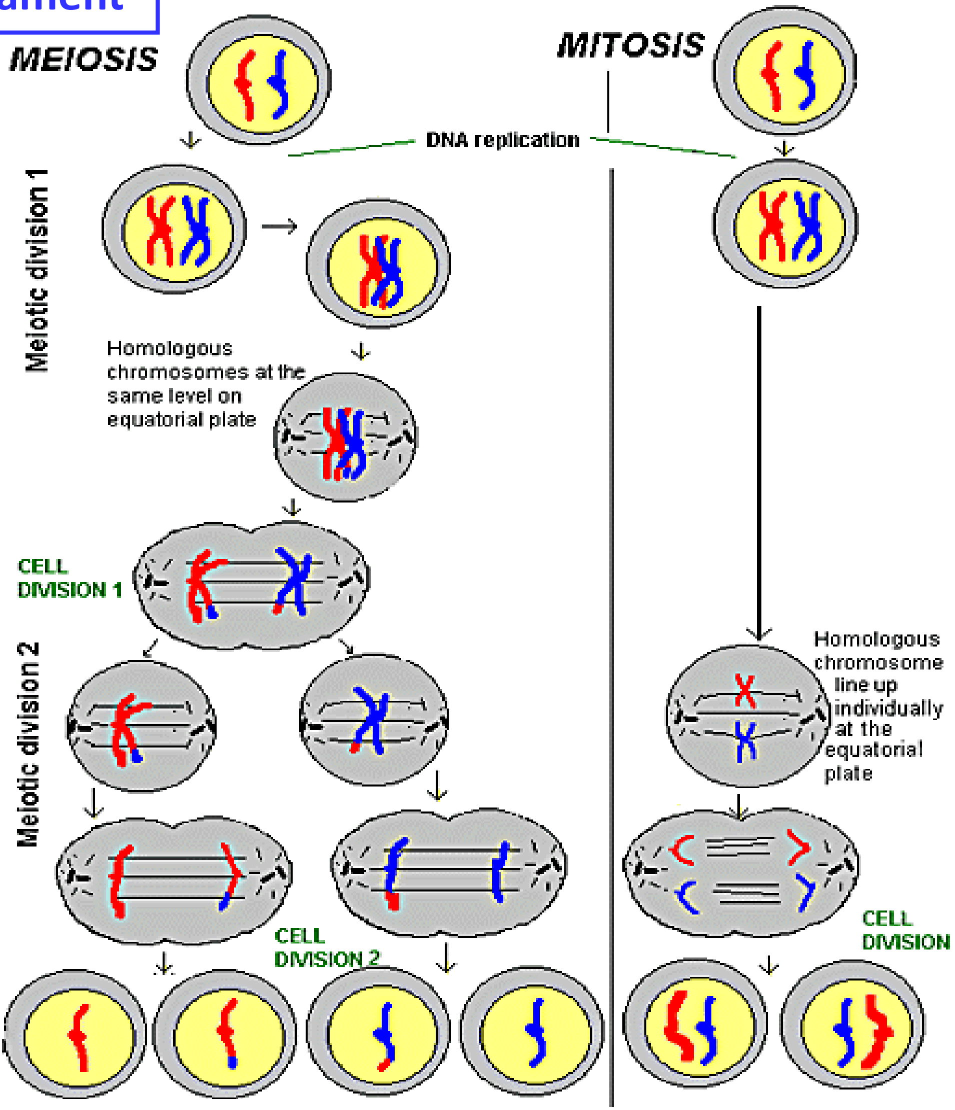
Basic Idea: Devise learning algorithms for complex environments that mimic effective adaptive and evolutionary processes found in nature.

Evolutionary Processes in Nature: Mitosis vs. Meiosis

Replication as in Axelrod Evolutionary Tournament

- Mitosis: one cell becomes two cells with the same DNA (cloning)
- Meiosis: one cell becomes four cells with one strand each (basis for sexual reproduction)

Permits "Genetic Evolution"!

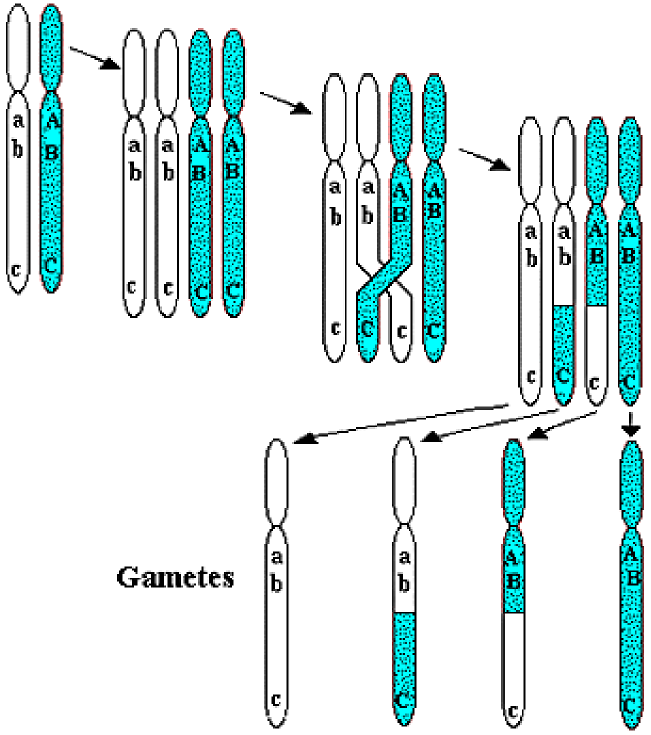

MEIOSIS
MITOSIS
DNA replication
Meiotic division 1
Homologous chromosomes at the same level on equatorial plate
CELL DIVISION 1
Meiotic division 2
CELL DIVISION 2
Homologous chromosome line up individually at the equatorial plate
CELL DIVISION

Crossover (Recombination)

- Meiosis -> production of germ cells

- Parts of two chromosomes get swapped.

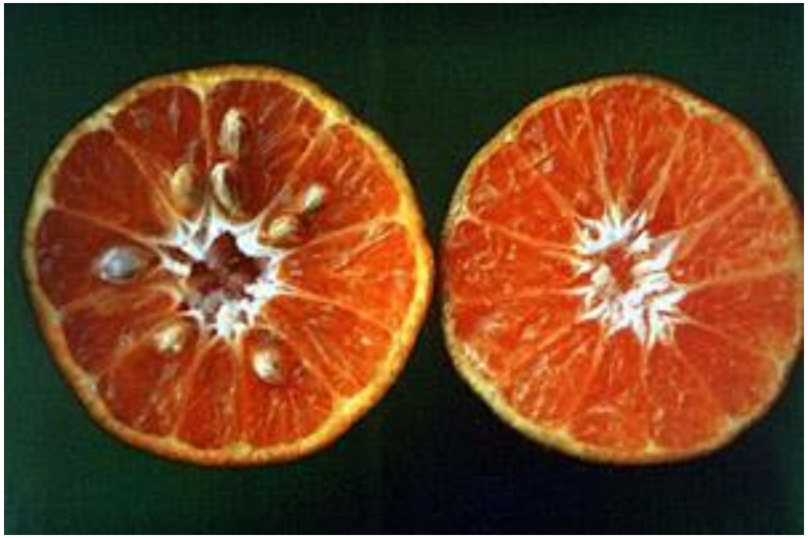
- Also called recombination



Crossing-over and recombination during meiosis

Mutation

- □ Occasional misfiring of the replication process.
- □ Almost always harmful.
- □ However, on occasion, it results in a “fitter” entity.



Differential Survival

- Once there is variability (through sexual reproduction, crossover, & mutation) in a population, the environment culls some members of the population while others survive.

- This process is termed ***Natural Selection*****.**

Evolutionary Learning Algorithm Example:
Genetic Algorithms (GAs)

- Directed search algorithm based on the mechanics of biological evolution

- Developed by John Holland, University of Michigan (1970's)

- **Original Goal:**

  To use adaptive and evolutionary processes found in natural systems as a metaphor for the design of ***effective search algorithms*** suitable for complex environments

Basic Steps of a Simple GA

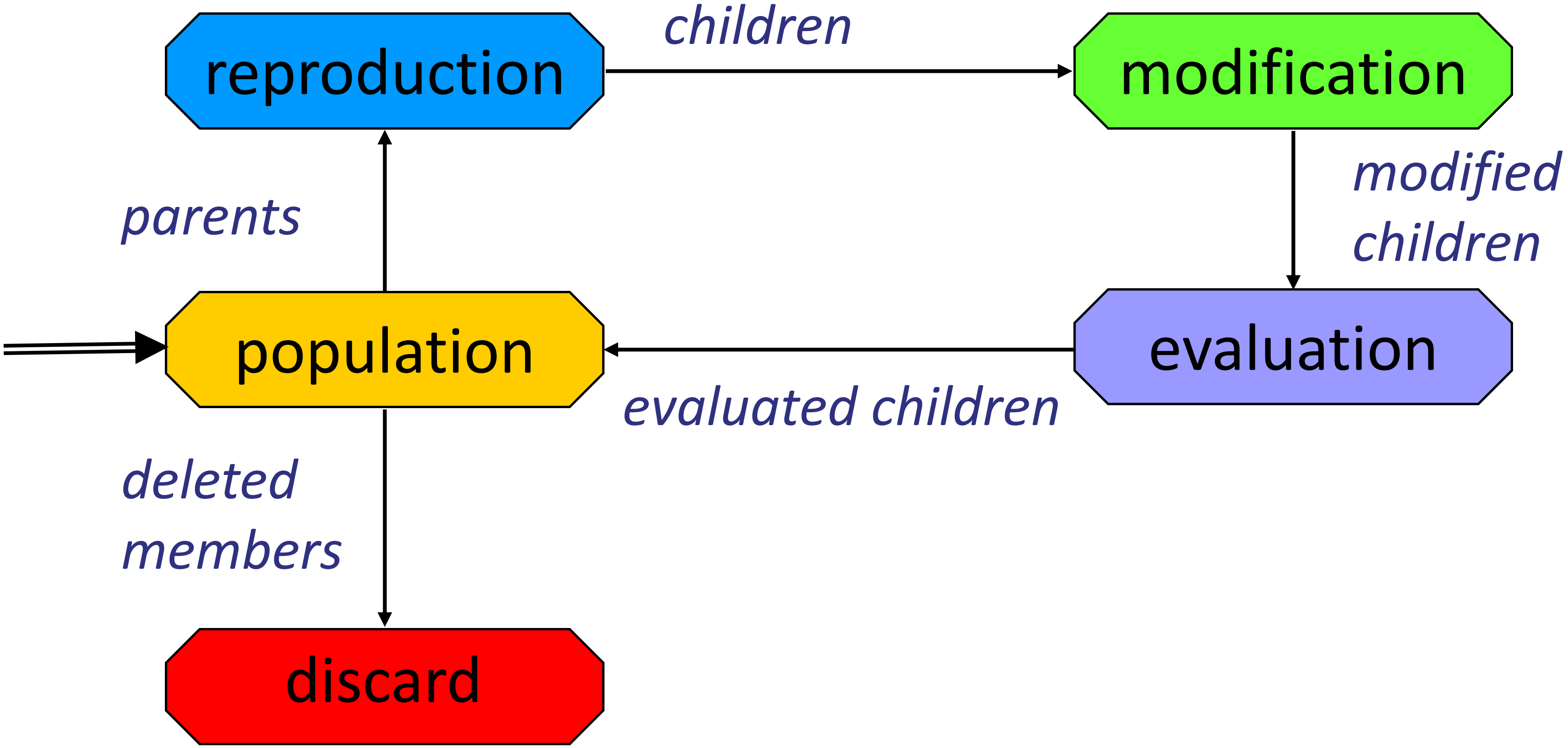
Step 0: Construct/configure an initial population of members (agents, strategies, candidate solutions to a problem, …).

Step 1: Evaluate the “fitness” of each member of the current population and discard least fit members.

Step 2: Apply “genetic operations”(e.g., mutation, recombination,…) to the remaining (parent) population to generate a new (child) population to replace discarded least-fit population members.

Step 3: Loop back to Step 1 and repeat.

The GA Cycle of Reproduction



What Might "Fitness" Mean?

EXAMPLES….

- The ability to solve a particular type of problem (e.g. a particular form of math problem)

- The ability to repeatedly perform some task (e.g., facial recognition)

- The ability to survive and prosper in some real or computational environment

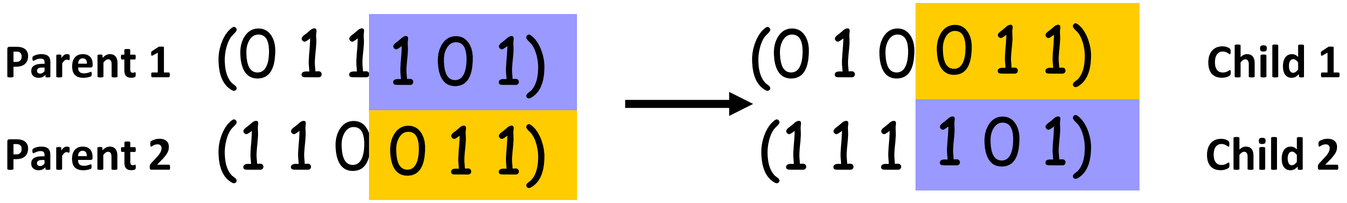
Representation of Population Members

EXAMPLE: Bit-String Representation (String of 0’s & 1’s)

- Population Members = PD Game Strategies
- One Possible Strategy **S**
  - State = (My last play, Rival’s last play)
  - Two Possible Actions: Cooperate=1, Defect=0
  - Four Possible States: 1=(1,1), 2=(1,0), 3=(0,1), 4=(0,0)
  - Strategy **S** = TFT:

    Start by choosing Action 1
    If State 1, then choose Action 1
    If State 2, then choose Action 0
    IF State 3, then choose Action 1
    IF State 4, then choose Action 0
- Bit-string representation of Strategy **S**: (1 | 1 | 0 | 1 | 0)

Crossover (Recombination)



Parent 1 (0 1 1 1 0 1) → (0 1 0 0 1 1) Child 1

Parent 2 (1 1 0 0 1 1) → (1 1 1 1 0 1) Child 2

Crossover is a potentially critical feature of GAs:

- It can greatly accelerate search early in the evolution of a population
- It can lead to discovery and retention of effective combinations (blocks, schemas,…) of S → A associations

Mutation of Population Members

Example: String Mutations

Before: (1 0 1 1 0)

After: (1 0 1 0 0)

Before: (1.38 -69.4 326.44 0.1)

After: (1.38 -67.5 326.44 0.1)

- □ Causes local or global movement in search space
- □ Can restore lost information to the population

Issues for GA Practitioners

- **Basic implementation issues**
  - Representation of population members
  - Population size, mutation rate, ...
  - Selection, deletion policies
  - Crossover, mutation operators
- **Termination criteria**
  - When is a solution good enough?
- **Fitness Function Specification**
  - “Solution” depends heavily on the fitness function (specification of “fitness” often the hardest part)

Types of GA Applications

Domain	Application Types
Control	gas pipeline, pole balancing, missile evasion, pursuit
Design	semiconductor layout, aircraft design, keyboard configuration, communication networks
Scheduling	manufacturing, facility scheduling, resource allocation
Robotics	trajectory planning
Machine Learning	designing neural networks, improving classification algorithms, classifier systems
Signal Processing	filter design
Game Playing	poker, checkers, prisoner's dilemma
Combinatorial Optimization	set covering, travelling salesman, routing, bin packing, graph colouring and partitioning

6. Connectionist Learning

Asks…

Does the learning of state-act associations s → a ("if s, then a") require a centralized information processor, or can it proceed through some form of decentralized information processor?

And can the appropriate specification of the conditioning states s be learned along with the appropriate specification of the associations s → a ?

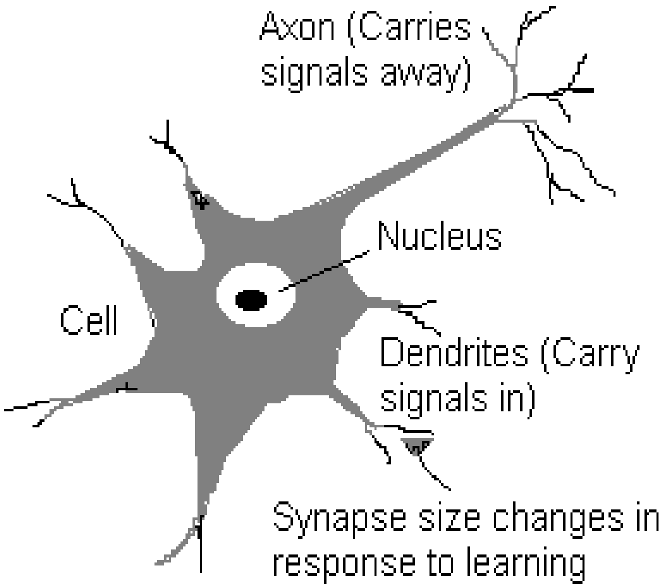
Connectionist Learning Example

Artificial Neural Networks (ANNs):

Decentralized information processing paradigm inspired by biological nervous systems, such as the human brain

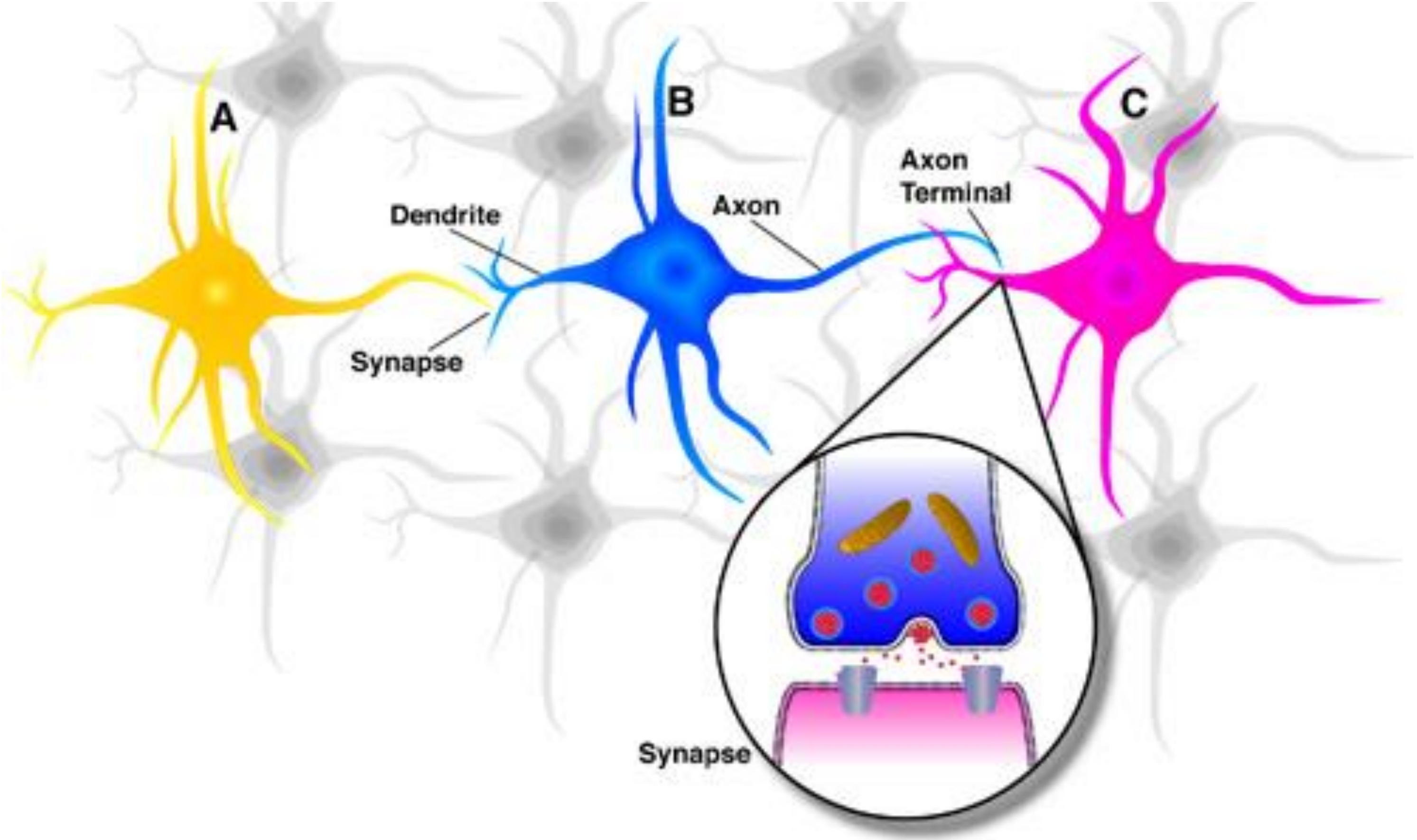
Inspiration from Neurobiology

- ***Neuron*** : A many-inputs/one-output unit forming basis of human central nervous system
- Output can be *excited* or *not excited*
- Incoming signals from other neurons determine if the neuron shall excite ("fire")
- Output subject to attenuation in the ***synapses*** (small gaps) that separate a neuron from other neurons at the juncture of its axon with their dendrites



Axon (Carries signals away)
Nucleus
Cell
Dendrites (Carry signals in)
Synapse size changes in response to learning

Connections Among Neurons



Structure of ANNs

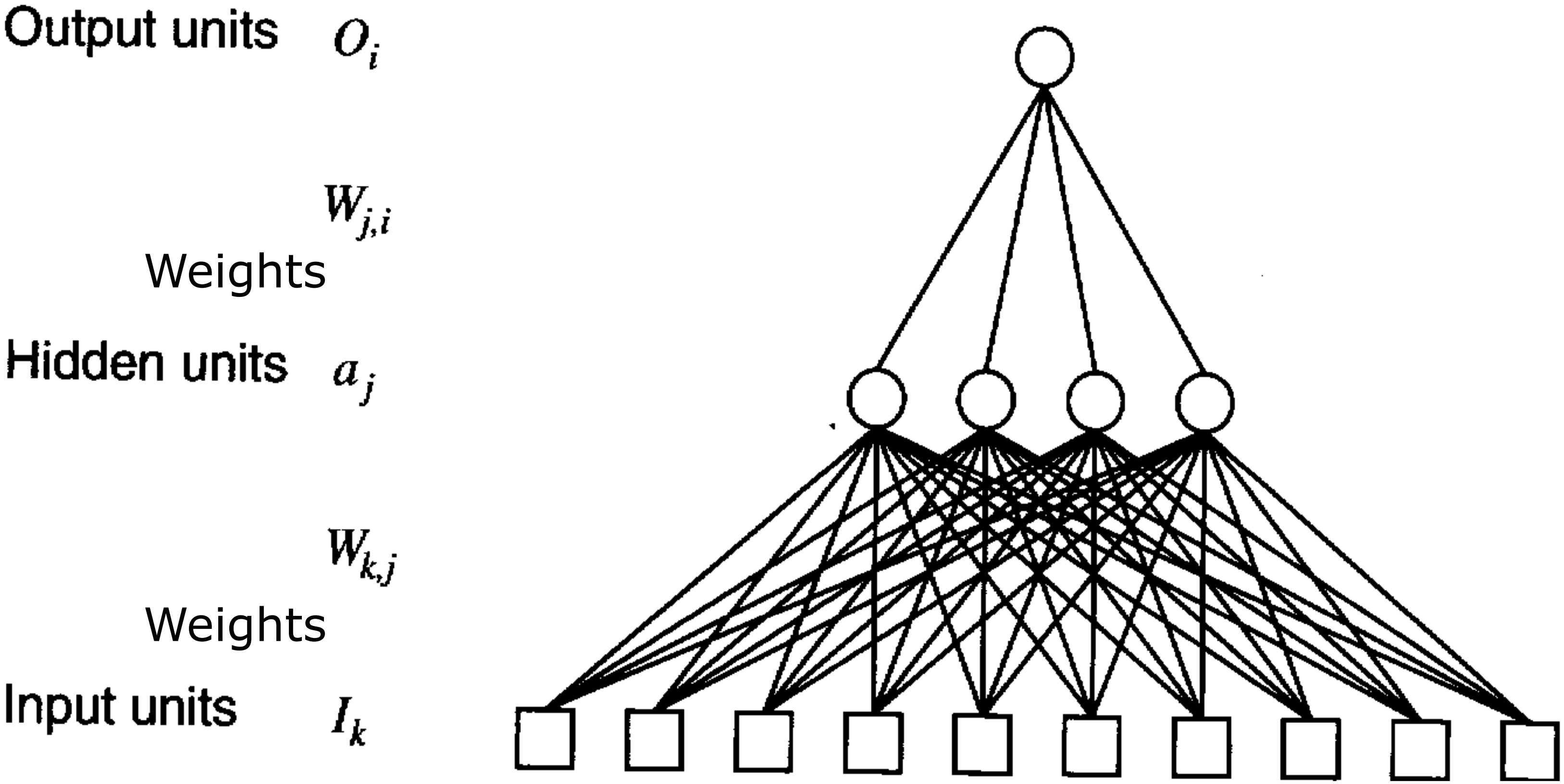
- Collection of interconnected processing units working together

- Structure = **(1)** Unit configuration (numbers of *input* units, *hidden* units, and *output* units); **(2)** Unit connections; & **(3)** Connection weights

- Structure can be updated via unsupervised learning, RL, or supervised learning

Example: Feedforward ANN

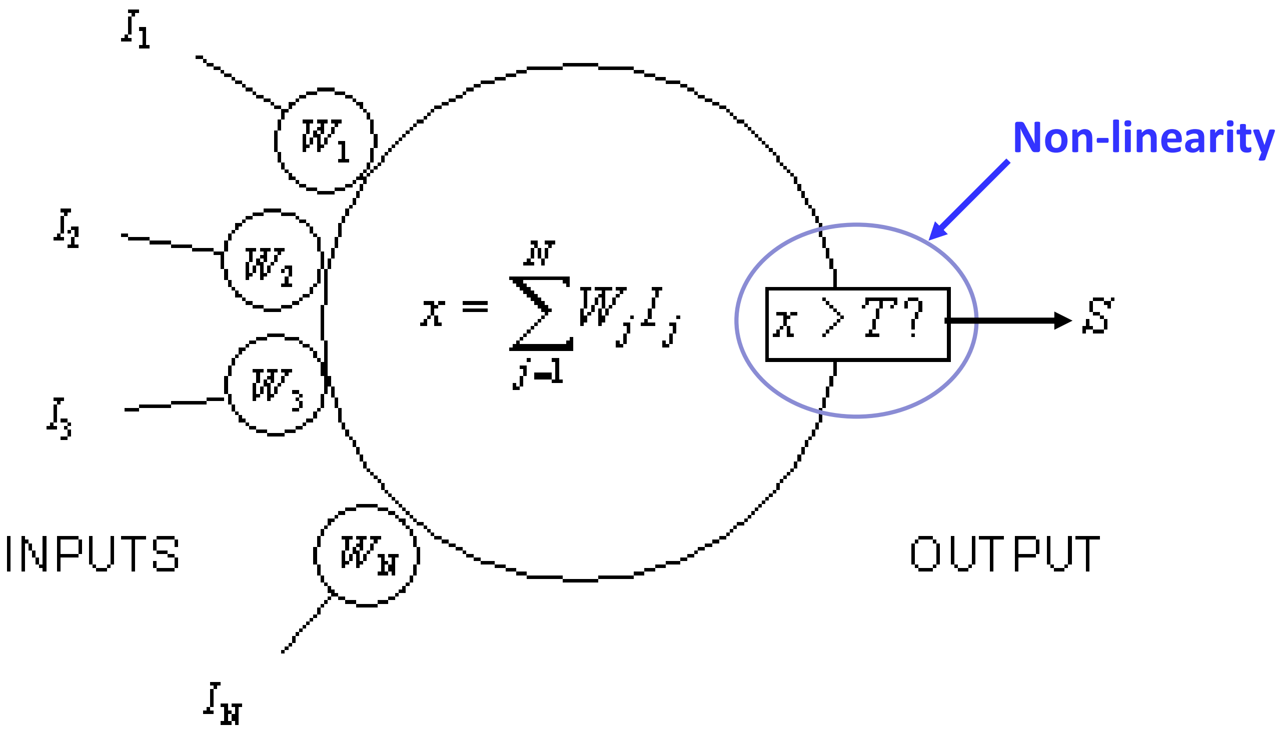
(No recurrent loops)



NOTE: **Here only one hidden layer is depicted. In general, a feedforward ANN can include multiple hidden layers, thus permitting deep(er) learning.**

Hidden Unit Representation

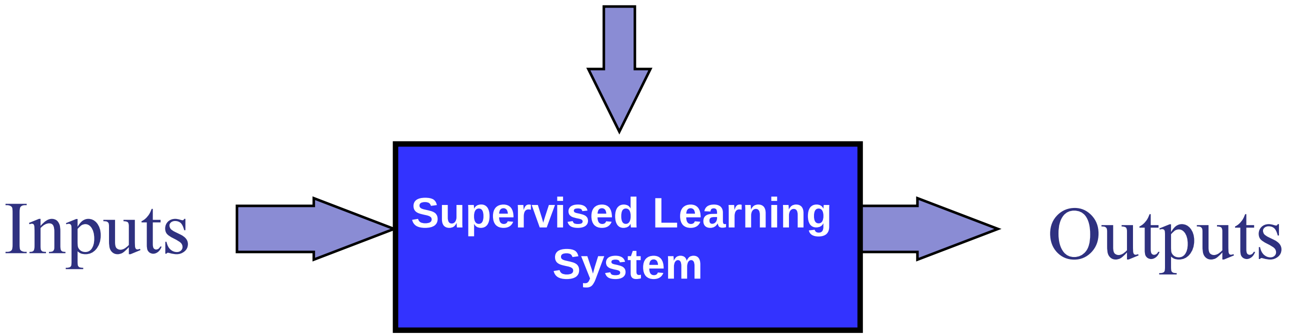
Example: The hidden unit depicted below calculates a weighted sum x of inputs I_j and compares it to a threshold T. If x is higher than the threshold T, the output S is set to 1, otherwise to -1.



ANN Supervised Learning

(Learn from a set of examples via *error-correction*)

Training Examples = Desired Input-Output Associations



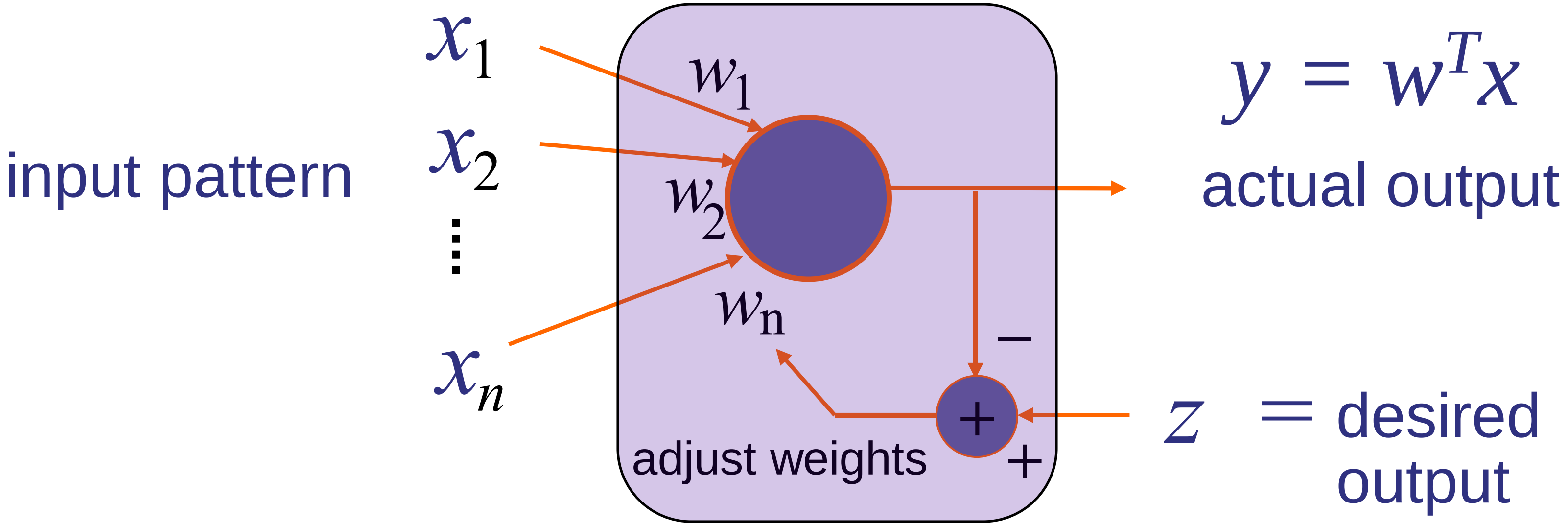
Error = [Desired Output – Actual Output]

ANN Supervised Learning via "Back Propagation"

- Desired input-output associations provided by supervisor through training examples
- ***Error*** = Difference between desired and actual output for any given input
- Weights updated relative to error size
- Start by calculating output layer error and weight correction, then "propagate back" through previous layers

Example: “Adaline” Learning Rule

Widrow and Hoff, 1960



input pattern

x_1

x_2

⋮

x_n

w_1

w_2

w_n

adjust weights

$-$

$+$

$y = w^T x$

actual output

z = desired output

$$\Delta w_i = \alpha[z - y]x_i$$

Illustrative ANN Applications

- **Prediction: Learning from past experience**
  - *Pick the best stocks in the market*
  - *Predict weather*
  - *Identify people with cancer risk*
- **Classification**
  - *Image processing*
  - *Predict bankruptcy for credit card companies*
  - *Risk assessment*

ANN Applications … Continued

- **Recognition**
  - *Pattern recognition: SNOOPE* (bomb detector in U.S. airports)
  - *Character recognition*
  - *Handwriting recognition (processing checks)*
- **Data Association**
  - *Identify scanned characters AND detect if scanner is working properly*

ANN Applications … Continued

- □ **Data Conceptualization**
  - *Infer grouping relationships*
    e.g., extract from a database the names of those most likely to buy a particular product.

- □ **Data Filtering**

  e.g., *Take the noise out of a telephone signal*

- □ **Planning**
  - *Evolve "best" decisions for unknown environments*
  - *Evolve "best" decisions for highly complex environments*
  - *Evolve "best" decisions given highly noisy input data*