IKT710

General Info:

No Lecture Friday

Monday:

9:15 – 10:30 Teach

10 min break

10:40 – 11:40 Teach

Lunch

13:00 to 14:30 Teach

10 min break

14:40 – 16:00

First assignment deadline Wednesday

Second assignment deadline Tuesday next week

# Monday:

# Expectations

You role a die, and get

1: 0.1, 2:0.1, 3:0.3, 4:0.2, 5:0.15 6:0.15 – Sum must become 1

{1+2+3+4+5+6}

All probabilities are 1 when summed

The average is essentially the occurrence divided by number of total throws, or 1/6.

E[x] x Zigma(1/1) x P(x=x)

1\*(100) + 2(100) + 3(300) + 4(200) + 5(150) + 6(150) = (100 + 200 + 900 + 800 + 750 + ..)/1000 = 3.65

# Law of the unconscious statistician

You role another die, and get

1:0.2, 2:0.2,… 5:0.1, 6:0.1

Average value is 3.1 (1\*200 + 2\*200 + 3\*200 + 4\*200…)

### Conditional Random variables

P(a|b) = P(ab)/P(b)

If B has no info about A.

P(A|B) = P(A) - Independent

P(a|b) = P(AB)/P(B) = P(a)P(b)

Y = Die that was rolled

Y = 1:0.2 2:0.8, 1 = Die 1(expectations) 2= die 2(Law of unconscious statistician)

## Total probability

Means P(x=x) = Sum y(P(y=x | y=y) P(y=y))

P(x=1) = P(x=1 | y=1)\*0.2 + P(x=1 | y=2)0.8 = 0.18

P(x=2) = 0.18

P(x=3) = 0.22

P(x=4)…

E[x] = 1\*0.18 + 2\*0.18… = 3.21

Knowing the average value of each die, and knowing how often a die occurs this can be used to calculate total average value. i.e. 3.65\*0.2 + 3.1\*0.8 = 3.21

Meaning that the E[x] = E[E[x | y]], the double expectations makes the condition disappear.

## Automaton: Finite state machine

Recognizer for 0 1 1

\* -0>\*-1> \*-1> \*

1 0 <-0 <-0 - stays or reverts

Last state is the accepting state

Finite state machine, contains states - memories, and transitions – computations, and output - choice.

## Learning Automata (LA) – in random environments

Teacher -> student

The teacher has many actions {alpha1, alpha2, … alpha r}

|  |  |
| --- | --- |
| Alpha | C – Penalty probability |
| Alpha 1 | 0.7} |
| Alpha 2 | 0.2} - Unknown |
| Alpha 3 | 0.9} |

At any time the student picks an alpha(action), the teacher responds with a reward or penalty called Beta. Beta 1 penalty, beta 0 reward.

Making the teacher the environment, while the student becomes LA.

Task

1. Design LA. – Must be automata

2. Learning in Random Environment

3. The set of Penalty probability are unknown

4. Analyse – What is a good machine / How to know what it is doing?

Analyses part requires markov chains

## Markov chains

Teach us:

1. What they are
2. How they perform
3. How to analyse them
4. ***How to simulate them***

Two ways to consider random variables:

* Independent
* Dependent – How does one capture this? (Markov – statistician)

## Dependent Random Variable

Make the current observation dependent of the past (one) observation.

Roll of a die + gambling result – Has 5 dollars

P – prob heads

1-P – Prob tails – q

The one who gets 5 wins and gets all the money.

State diagram is the markov chain.

Mathematical tool – Matrixes. Markov chain must be converted to matrix.

0[ 1 0 0 0 0 0]

1[ q 0 p 0 0 0]

2[ 0 q 0 p 0 0]

3[ 0 0 q 0 p 0]

4[ 0 0 0 q 0 p]

5[ 0 0 0 0 0 1] – absorbing state

Has 2 diagonal matrix, and 2 absorbing ones

States are called phi

When Mij = P[state(n+1) | state(n)]

P[phi(n+1) = phi j | phi(n) = i]

Sum j(Mij) = 1, when you go from one state you must go to another

Mij >= 0

### Properties of markov chains

Quantity called pi – Pi i(n+1) = P[phi(n) = phi i]

=sum j( P[phi(n+1) = phi I | phi(n) = phi j] \* P[phi(n) =phi j])

Pi j(n+1) – in terms of j pi

Phi(n) – state at time n

Pi i(n) – probability that the state of phi(n) = phi(i)

At time n what is the probability that the machine or markov chain is state i.

Pi i(n+1) = sum j[phi(n+1)=phi I | phi(n) = phi j] \* P[phi(n) = phi j]

How could I be in state I at time n+1 – total probability conditional probability

Scalar (pi i)

[pi 1(n)] [m11, m12, …, m1s] [pi 1(n+1)]

[pi 2(n)] [m21, m22, …, m2s] [pi 2(n+1)]

…

[pi s(n)] [ms1, ms2, …, mss] [pi s(n+1)]

What is pi(n)

Pi(n) = Mt\*pi(n-1)

= Mt\*Mt\*pi(n-2)

Pi(n) = (M\*\*3)t \* pi(n-3)

What is pi(n)?

(M\*\*n)t \* pi(0)

General equation for a MC

Pi(n+1) = Mt pi(n)

An ergodic chain is a non-absorbing markov chain, they all converge meaning that

Pi(infinity) = Mt\*pi(infinity) – this is called an asymptotic/stationary/equilibrium distribution

Eigenvector equation

#### Communicating classes

State I state j

1. Accessible. [i]->[j] – J is accessible from I.
2. Communicate [i]->[j] and [j]->[i] – I and J communicate

All the states in a MC, form closed communicating classes.

I🡨🡪j j🡨🡪k 🡺 i🡨🡪k

Periodicity of 1 – How often can you get back to where you were, once you leave.

Greatest Common Divisor (GCD)}Ergodic

State 1 can go to state 2, which can go to state 3, which can go to 1 or 4, 4 can go to 2,3 and 1.

This means there are essentially unlimited ways of getting back to state 1, meaning that the gcd is 1

When pi goes to infinity, only the constants remain as we use probabilities, and 0.something\*\*infinity is always 0, so it no longer matters.

Need to find psy 1, psy 2, .. psy infinity, and lambda 1, lambda2 , lambda infinity

For an ergodic M lambda 1= 1 lambda 2, … lambda s<1

Therefore:

1 pi(infinity) = Mt\*pi(infinity) = eigenvector equation

The eigenvalue for lambda is 1

Pi\* = pi(infinity)

Pi\* = Mt\*(Pi\*) ==> Pi \* = (M\*\*infinity)t\*pi(0)

(M\*\*n) only needs to be done 6 times, as this would give M\*\*64, which is a high number given that it converges.

### Absorbing MC

Has absorbing barriers, like winning all the possible money, or loosing all your money in a casino. All non-absorbing states are called transient states. What is the probability of converging to each absorbing state.

fiA – probability of converging to A if I start from i.

***Chapman. Kolmogorov equations***

How can I go from I to a?

fiA = MiA + sum j(Mij) fjA – In any number of steps

[f1A f2A … fJA] = [m1A m2A … mJA] + [m11 m12 … m1J, m21 m22 m2J, … mJ1 mj2 ... mJJ] [f1A f2A … fJA]

Do not transpose like with ergodic

[f1a -> fJA] – [mJJ] = [m1A -> mJA

[I-Mtau][f1A… fJA] = [m1A… mJA] solve

F1a… = [I-Mtau\*]inverse[m1A… mJA]

I = [1,1,1] (identity matix)

Mtau = the transient probability

[1 -0.7 0, -0.2 1 -0.6, 0 0.1 1]inverse \* [0, 0, 0.9]

Similarly you can do the same for ending up in state 1 where you cannot get out.

### Simulation

1. Choose and event from a distribution

[p1 p2 p3 p4] = [.1 .4 .3 .2]

1 2 1 3 3 4 2 1 … 🡪 simulate

Compute Cumulative Distribution

0-0.1=1, 0.1-0.5=2, 0.5-0.8 = 3, 0.8-1 = 4

The simulation should use binary search, as linear search is bad for large number of actions

Simulate MC

Input:

M = [\*]

Pi(0) or starting distribution

Output:

2 1 3 4 2 1 1 4 3 2 …

## Result Ergodic chain

E1 3 1 2 1 4 2 1 3 …

E2 1 4 2 3 ...

E3 3 2 4 1 …

E100 2 1 3 1 4 2 …

Each experiment should have say 10000 numbers, which simulates a long MC

## Absorbing MC

Time average is meaningless

Ex 1 -> 1 (2 – 1 – 1 – 1 – 1 – 1 -… 1)

Ex 2 -> 3 (2 -3 -3 – 3 – 3 – 3 – 3)

Essentially run it till it is absorbed, then stop. Run it for many episodes.

# Tuesday:

## Difference equations

Like differential, but discrete

Xt = AX(t-1) – Recursive definition

What is the closed form expression for Xt?

Solution: Xt = Aa\*\*t

Proof:

If the sol is true

Xt = Aa\*\*t

X(t-1) = Aa\*\*(t-1)

Xt/X(t-1) = Aa\*\*T/Aa\*\*(t-1)

Xt/ X(t-1) = a => Xt = a(t-1)

MC:

Any markov chain is a vector form of a diff equations

Xt = A X(t-1)

Closed form expression for Xt

1. X(inf.) = AX(inf)🡺 eigenvector lambda = 1
2. Expression for any time t

Pi i(t) = A1 + A2lambda2\*\*t + A3lambda3\*\*3… AsLambdas\*\*s

Steady state – Time goes to inf. Depends on lambda = 1

Transient – Depends on the other terms (A2Lambda2\*\*t…) and it drops off depending on the magnitude of lambda(i’s)

## Foundation of LA

Given an env and an agent, which have a feedback loop. The agent chooses an action, while the env gives a penalty probability. The environment has some actions, and some penalty probability, which can be either 0 or 1. We have no control over the environment. We design the LA (Agent) (Think ikt440 dam optimization). The agent has a set of states (memory), a set of possible actions (decision), input from the environment (beta(0 or 1)), the update of the memory (changes to the memory)(F), how to decide (G).

### What is learning?

R - actions

P – probability of choosing the actions at time n. Probability that at time n the action is alpha i. The sum of all these probabilities = 1.

If it is being chosen based on the sum, what is the penalty at time n?

M(n) – average penalty at time n = r Sum i(CiPi(n)) where Ci is penalty probability and Pi is probability of choosing the actions a time n.

The goal is to minimize m(n)

M(n) = C1P1\*\*(n) + C2P2 + … CrPr\*\*(n)

Assume that Cl is the lowest

M(n) = probability of reward given action n \* prob of choosing an action at time n + same for n+1

If Cl is the smallest probability of punishment, then the probability of choosing l should be 1. As it is the best action to take. Which makes the probability of choosing the other actions 0. This assumes that we have tested this for a longer period of time, which makes the reward probabilities converge.

We know there is an optimal solution.

LA is optimal when time goes towards infinity and the loss reaches the final solution. i.e. you attempt the different actions, and as you are told what the best is, you do it more and more. Think of Deep learning and momentum.

There is unfortunately no optimal LA with finite memory.

LA is converging optimally

There are many Epsilon optimal learning automata, meaning it converges to the optimal.

### Least we can expect

Choose all the actions randomly (wp(1/R)

M0 = R Sum i(Ci (1/R) = (1/R) sum i(Ci)

If n-> inf M(n)<M0 – LA is expedient (Try to at least beat random, as there is no point if not)

### Absolutely Expedient – Continuously learning

You learn continuously, meaning that you will become better/smarter for every step. Monotonic learning.

## Designing Learning Automata (fixed structure)

1. Memory - What to remember, How to remember it, How to update the memory
2. Strategy – How to update the memory, how to make a decision

This should be done using mathematical tools, via matrices.

1 is done using a matrix called F, and 2 using a matrix called G

#### F Matrices – How to update memory

How much memory do we have? N – Notation

Ex: N=4, meaning 4 states, R=2, meaning 2 actions. Whenever in state 1 or 2 choose alpha 1, otherwise choose alpha 2. If alpha 1 gives penalty, move to state 3 or 4, and opposite. This leads to G [[1,0], [1,0],[0,1],[0,1]].

F\*\*0 – F reward

One example is if alpha 1 means stay on the left side, and alpha 2 means stay on the right side. By staying on the right side, we mean to loop back into the same state.

Making F reward:[[1 0 0 0], [0 1 0 0], [0 0 1 0], [0 0 0 1]]

F\*\*1 – F penalty

Whenever you choose an action which gives you penalty you move to the other side. From 1 you go to 4, from 2 you go to 3 and opposite.

Making F penalty:[[0 0 0 1],[0 0 1 0],[0 1 0 0], [1 0 0 0]]

F reward is the F given reward, and F penalty is F given penalty.

#### Stochastic transitions

F reward means that if you are at state 1, you move to state 2 with a probability of 0.2 and stay at state 1 with a probability of 0.8. The same goes for state 2 though 0.1 instead of 0.2 and 0.9 instead of 0.8. The highest probability here is the probability of staying. The same is true for state 3 and state 4, with state 3 acting like state 2, and state 4 acting like state 3.

F0 = [[0.8 0.2 0 0],[0.1 0.9 0 0],[0 0 0.9 0.1],[0 0 0.2 0.8]]

F penalty if state is 1, move to state 2 0.1, state 3 0.9, from 2 loop with 0.2, and 0.8 state 3, if the state is 3, 0.8 to move to state 2, 0.2 to loop, from 4 move to 3 0.1 and to 2 0.9.

F1 = [[0 0.1 0.9 0],[0 0.2 0.8 0],[0 0.8 0.2 0],[0 0.9 0.1 0]]

Both F0 and F1 can be both deterministic and stochastic.

G can also be stochastic

If alpha 1 = 0.9/0.8 at state 1 or 2, alpha 2 = 0.1/0.2

At state 3 or 4 alpha 2 = 0.8/0.9, alpha 1 = 0.2/0.1

G [[0.9 0.1],[0.8 0.2],[0.2 0.8],[0.1 0.9]]

F I,j \*\*beta = P[phi(n+1)=phi(j)| phi(n)=phi(i); beta is input] Sum over j must be equal to 1

G I,k = P[alpha n = alpha k | phi(n) = phi (i)] Sum for all values of k must be equal to 1

Without loss of generality G can always be deterministic

You give M1: F1 rew, F1 pen, G1 this is stochastic

I give M2: F2 rew, F2 pen, G2 this is deterministic

G after this point is deterministic

Composite F Matrix ~F(F tilde)

F rew = [[],…,[]] F pen = [[],…,[]]

Ftilde = Fij rew[1-c2] + Fij pen[1-c1]

Design: Requires F0, F1 and G which is always deterministic

Analysis: From F0 and F1 get Ftilde

Solve Ftilde Pi\* = Pi\* - Probability of being in the state

P1\*, P2\*

### Tsetlin Machine

He designed Linear Tactic L2(number of states),2(number of actions)

F0 = [[1 0], [0 1]]

F1 = [[0 1], [1 0]]

Ftilde = [[1-c1 c1],[c2 1-c2]]

C2/C1+C2

C1/C1+C2

L4,2 – 4 states 2 actions

The order is not 1,2,3,4 but instead 1,2 4,3

On reward:

State 1 to 1, state 2 to 2, state 4 to 3, state 3 to 3

On penalty:

State 1 to 2, state 2 to 4, state 4 to 2, state 3 to 4

The order is very important for the analysis

F0: [[1 0 0 0], [1 0 0 0], [0 0 1 0], [0 0 1 0]]

F1:[[0 1 0 0],[0 0 0 1],[0 0 0 1],[0 1 0 0]]

Ftilde = [[1-c1 c1 0 0],[1-c1 0 0 c1],[0 0 1-c2 c2], [0 c2 1-c2 0]]

P1\* = pi1\*+pi2\*

P2\* = pi3\* + pi4\*

Solution = [c2\*\*2/(c1\*\*2+c2\*\*2) c1\*\*2/(c1\*\*2+c2\*\*2)]

Tsetlin L(2N,2)

F0 = [[1,0, 0,…,0][1, 0…, 0][0 1 0, …,0]…[0,0,…0,1]

F1 = [[0 1 … 0][0 0 1 … 0]…[0 … 0 1]]

# Wednesday

L2n,2 is epsilon optimal only when Cmin < 0.5

Theory of learning says you must treat rewards more seriously, if the reward and penalty is treated the same it does not work.

Krinsky, krolow…

Krinsky machine is epsilon optimal for all Cmin

### Simulation

Simulate both environment and agent

#### Tsetlin

The environment needs an action, number of actions, state and reward/penalty.

The agent just gives the action, and gets a reward/penalty, acts based on it.

For the starting position should be at the middle N or 2N

### How general ca we get?

F0, F1 = once fixed – the LA is fixed

Varshawskii & Vorenstora (V & V)

Why keep F0 and F1 fixed? Why not keep changing them

F0(time 0) F1(time 1) 🡺 F0 (time 1) F1(time 1)🡸🡺F0(time n) F1(time n)

Leads to F0 and F1 having n\*\*2 elements (NxN size)

Problem How to determine F0(N), F1(n) – Machine change, also how will you analyse them?

F0(time 0) and F1(time 0) gives us Ftilde(time 0)… F0(time n) F1(time n) gives us Ftilde(time n)

Pi(1) = Ftilde(0) Pi(0)

…

Pi(n) = Ftilde(n-1) Pi(n-1)

### V & V

Any variable structure SA (VSSA) is equivalent to an action probability updating rule.

VSSA F0 F1 updated is equivalent to P(n) -> P9(n+1), forget about states, and choose the actions based on a probability P(n)

P(n+1) 🡨 H(P(n),action(n),penalty(n))

Assume you have 2 actions - use a linear function – Linear multiplying constant (set it to 0.8 for reward, and 0.9 for penalty)

[0.5 0.5] choose action 1 or action 2 randomly, you get either a reward or a penalty. If you get a reward increase the probability of the action which gave a reward e.g. [0.6 0.4]. This should be done by multiplying the one which did not give a reward by the constant, and setting the other one to 1-other action. If there was a penalty with the matrix [0.5 0.5] this would lead to [0.45 0.55] given action 1, as the penalty means multiply with 0.9

Previously P1 has been a number, though here it is a random variable.

When n=1 there are 4 possible probabilities, this is because depending on the action the probability can be changed in different ways. When n=2 there are 16 probabilities, this increases in the same way 4\*\*n-1. meaning n=m -> 4\*\*m probabilities. When n=inf. The final distribution is increasing to middle, then decreasing.

If there is no penalty change. Then this number of different probabilities increases by 3\*\*n. Though when n goes towards infinity there exist only 2 probabilities.

## Types and updating rules

Linear or non-linear

Process rewards/penalties or both.

Lri – Linear reward inaction

Lip – Linear inaction penalty

Lrp – Linear reward penalty

Lrp and Lip gives you ergodic schemes, and Lri gives you absorbing schemes.

## Method of analysis

There always exist an updating rule

It updates of the form P(n+1)🡨H(P(n),alpha(n),beta(n))

Lrp 🡪 1 value 🡪 4 values 🡪 16 values... consider Estimation(P1(n+1)|P(n))

E(P1(n+1)|P(n))

Take Expectations again 🡺 E(P1(n+1)) = function of expectation(P1(n))

## In general

Pj(n+1)🡨Pj(n) - gj(prob(n)) alpha i, beta = 0

where gj(prob(n)) is a function for how much to decrease the pj by.

Given 4 actions:

[.4 .1 .2 .3] -> [.32 .08 .16 .44] – decrease everything by lambda or 0.8 in this scenario, then make the last one 1-sum of the others.

## Ergodic schemes

Converges to a distribution

A and b are the constants of how much something is changed, if getting a reward or penalty respectively.

To get the average sum for the distribution, do kp1\*P2(1-C2)+(1-KP2)\*(P1(1-C1)) and so on...

Also P2 = 1-p1, so this can be rewritten to not contain P2.

KP1 \* (1-p1)(1-C2)+(1-k(1-P1))\*P1(1-C1)...

Which will eventually give us the average value of the distribution.

Moral:

Lrp scheme (2 actions)

If Kr = Kp this is useless, as it will act like a tsetlin machine. 🡺 Kr<Kp<1, meaning the reward changes the values more than the penalty.

How to get the Epsilon optimal?

Meaning how do you choose Kr and Kp? Kr being 1-a, and Kp being 1-b

Kr-->1

Kp>>Kr->1

Kr must never be equal to Kp

R – Action Lrp

A, b – general case

Expectation[P1(n+1)|P] = kp1\*P2(1-C2)+(1-KP2)\*(P1(1-C1))...

## Thursday

### Ergodic schemes (linear)

Lrp scheme – Kr (a) Kp(b) a=b

Penalty = reward – which is bad

C2/(C1+C2) C1/(C1+C2)

Pi\* = (1/Ci)/(C sum j=I (1/Cj))

Kr<Kp<1

Kr-> 1

Kp 🡪 🡪 1 – faster than Kr

As Kr goes to 1 and Kp goes faster to 1, you get arbitrarily close to 1 in accuracy

### Absorbing schemes

Lri – Scheme – 2 actions

[0.5 0.5] Kr = 0.9 Kp = 1

Reward, decreases the other action by 0.9 (val\*0.9)

Penalty, means no change

Given enough iterations, you get either [1,0] or [0,1]

If C1< C2 converge at [1,0]

Else converge at [0,1]

These schemes are absolutely expedient, they are able to learn monotonically.

Meaning that if c1<c2 Expected[P1(n+1)] > Expected[P2(n+2)]

For a static env, use absorb. And ergodic for variable envs.

#### Property of Lri scheme

Pi(n+1)<- K (since Kp is 1, there is only Kr)

If action 2 is chosen and you get a reward (beta=0)

i.e [0.6 0.4] -> [0.54 0.46] when k=0.9

1-Kp2 | a1(action 1), reward [0.6 0.4] -> [0.64 0.36]

If there is a penalty, do not change probabilities

P1(n+1) – P1(n) 🡨 KP1 – P1 = -P1(1-k) (1-k is always positive)(-P1 is always +) with prob P2 \* D2

🡨 1-KP1 – P1 = P2(1-k) with prob P1 D1

🡨 0 with prob P1C1 + P2C2

Meaning action 1 with reward, action 2 with reward, and either with penalty

Meaning expected value = -P1(1-k)P2D2 + P2(1-k)P1D1 = P1P2(1-k)[D1-D2]

If a1 is better, then D1>D2

Take Expectations again

E[P1(n+1)-P1(n)] > 0 – For the better action P will monotonically increase.

What is M(n) = C1P1(n) + C2P2(n)

E[M(n)] = C1 E[P1(n)] + C2 E[P2(n)] – if the P1 increases, P2 decreases. And vice versa.

Lri scheme – 2 actions

Change(P1) > 0 for the best action

Change in loss M(n) < 0 for all n

R – Action Lri

[0.4 0.3 0.1 0.2] 🡪 a1 chosen, reward [0.46 0.27 0.09 0.18] as action 1 was good.

P1(n+1)🡨kP1 if any action is chosen, reward

🡨1-sum j(kPj) meaning you increase only the good action, which is increased by as much as the others are decreased.

P1(n+1) – P1(n) 🡨 KP1-P1 = -P1(1-k) with prob sum j!=1(PjDj)

1-sum j!=1(kPj) = (1-P1)-k sum = (1-P1)(1-k) with prob P1D1

E[P1(n+1)-P1(n) | P] = -P1(1-k) Sum j!=1(PjDj) + (1-k)(1-P1)P1D1

(1-k)[-P1 Sum j!=1(PjDj) + P1 Sum j!=1(PjDj)]

(1-k)[P1 Sum j!=1(Pj(D1-Dj))] if a1 is best action

When we stated

Pj(n+1) 🡨 Pj(n)-Gj ai, rew

🡨Pj(n) + sum i!=j(Gi) aj, rew

Meaning for action j, you increase its probability if it gets a reward. And you decrease the probability for action j if you get a reward for another action.

Any scheme obeying Ng(X) is … Exponential If and only if:

G1/P1 = G2/P2, …, Gr/Pr = lambda (arbitrary function)

H1/P1 = H2/P2, …., Hr/Pr = mew (arbitrary function)

Symmetry Conditions

G1 = lambdaP1 H1 = mewP1 if lambda = a = 1-k and mew = 0 this means Lri scheme

G2 = LambdaP2 H2 = mewP2

… …

Gr = Lambda Pr Hr = mewPr

Example:

lambda=a+b(P1P2P3…Pn) a=0.1 b=100

mew = C (P1P2P3…Pn)

if any Pi = 0 -> no change

## Discrete LA

FSSA (Tsetlin, krylov) Fixed – 10000 iterations are typical for convergence

VSSA (Lri) Variable – 1000 iterations are typical for convergence

The problem with VSSA is that they will not get to say [1 0], only very close to it say [0.9999 0.0001]

We do not know all of the values between 0 and 1 only N of them, say 10. This would mean that we only know 0, 0.1, 0.2,… 0.9, 1.0.

2 actions [0.5 0.5]

DL

Ri

Rp

Ip

If a1 and rew, then [0.6 0.4], if a2 and rew [0.4 0.6]… meaning you are only making discrete probability jumps.

DLrp N=2

You only know 3 states, 0 0.5 and 1.

ADLrp – Epsilon optimal for all envs (Absorbing DLrp)

### Pursuit Algorithms

Essentially go towards the action which gives you a reward. But in a straight line.

# Project

Goore game – Try to find the top of a unimodal distribution without talking to each other. If this is at 0.6, then 0.6 should say yes for them to get a reward. The chance of reward is the height at the percentage. Meaning if .6 is 0.75, if .6 say yes then there is a 75% chance of reward.

Environment can give some feedback to determine it