Lecture 39 – final exam review

Mark Hasegawa-Johnson

5/4/2020

Facts about the final exam

- Final exam will be on Compass.
- The regular section is 7-10pm on May 13.
  - Alternate exam is 7-10am May 13; if you prefer 7am, please e-mail the instructor.
- Exam is open-book, open-notes, open-internet. You can search for a solution on-line, but you MAY NOT ASK FOR HELP from another human being.
- Exception: you can ask the instructors, if you have a question. Piazza will be locked so that it ONLY accepts private posts to the instructors; please use piazza if you have questions during the exam.

Topics covered

There will be approximately 20 questions on the exam (at most two parts per question), distributed as follows:

- Part 1 of the course (Search-based AI): ~4 questions
- Part 2 of the course (Probability-based AI): ~4 questions
- Part 3 of the course (Learning-based AI): ~12 questions

Material from part 1 of the course

- Search (BFS, DFS, UCS, Greedy, A*): lectures 2-4
- Constraint satisfaction problems & planning: lectures 5 and 8
- Robots/configuration space: lectures 6-7
- Two-player games (minimax and alpha-beta): lectures 9-10

Material from part 2 of the course

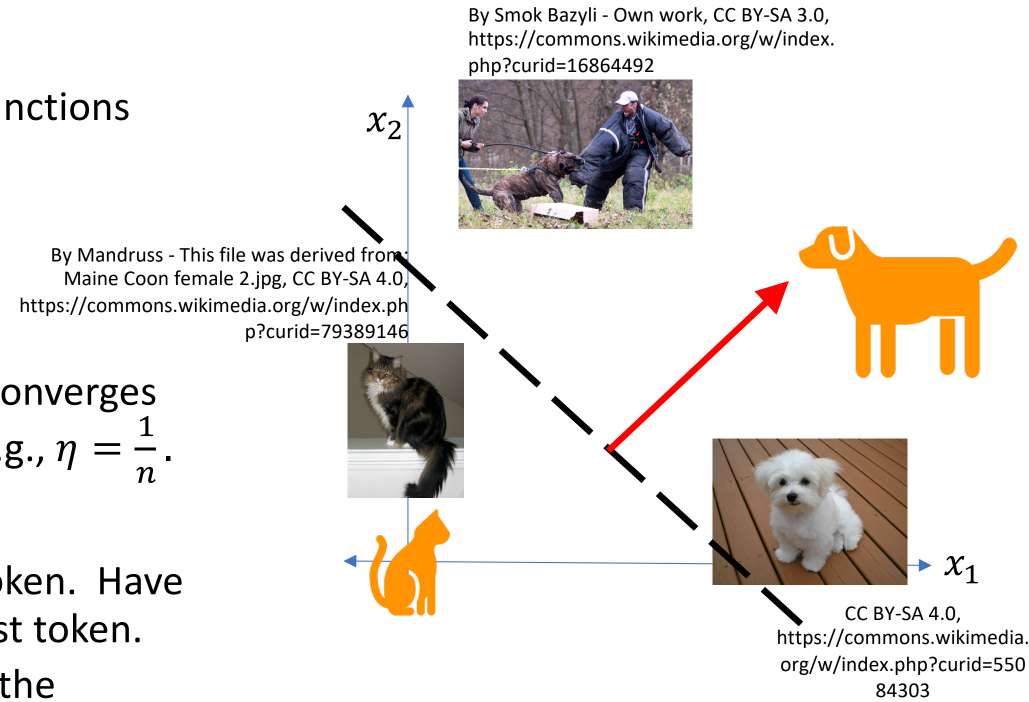
- Probability & Naïve Bayes: lectures 12, 14
- Bayes Nets: lectures 15, 16
- Natural language and Computer vision: lectures 17, 20
- HMMs: lectures 18, 19

Material from part 3 of the course

- Linear classifiers, KNN, Perceptron: lectures 22, 25, 26
- Differentiable loss functions & Deep neural networks (logistic regression, softmax, cross-entropy loss, back-propagation): lectures 26-28
- MDP (Bellman's equation, value iteration, policy iteration) & Reinforcement learning (model-based, Q-learning, deep Q-learning, actor-critic): lectures 29-32
- RL for two-player games, games of chance, imperfect information: lectures 33-34
- Game theory & Mechanism design: lectures 35-36

Linear classifiers

- $y^* = \mathrm{sgn}(w_1 x_1 + \cdots + w_D x_D + b) = \mathrm{sgn}(\vec{w}^T \vec{x})$
- A linear classifier can learn many types of binary functions (e.g., AND, OR, NOT), but it can't learn XOR.
- **<u>Perceptron</u>**:
  - If $y_i = y_i^*$ then do nothing.
  - If $y_i \neq y_i^*$ then set $\vec{w} = \vec{w} + \eta y_i \vec{x}_i$
  - If the data are linearly separable, perceptron converges with $\eta = 1$. If not, you need a decreasing η, e.g., $\eta = \frac{1}{n}$.
- **<u>K-nearest neighbors (KNN)</u>**:
  - Find the K training tokens closest to the test token. Have them vote: majority label is the label of the test token.
  - Result is a piece-wise linear classifier because the boundary between the features where training token #1 is closest, vs. training token #2 closest, is a line.



Deep neural networks

- Each **excitation** is a linear combination of the previous node's activations:

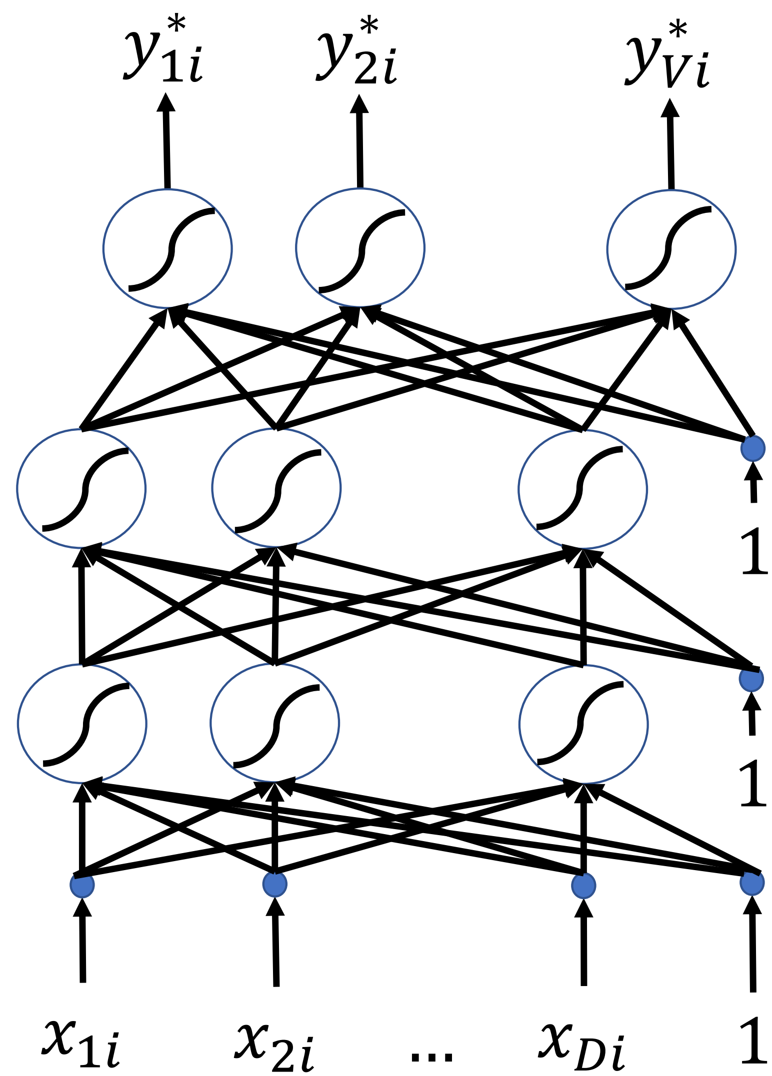
$$\beta_{ki}^{(l)} = \sum_{j=1}^{N+1} w_{kj}^{(l)} h_{ji}^{(l-1)}$$

...where $w_{kj}^{(l)}$ is called a "network weight," and will be learned using back-propagation.

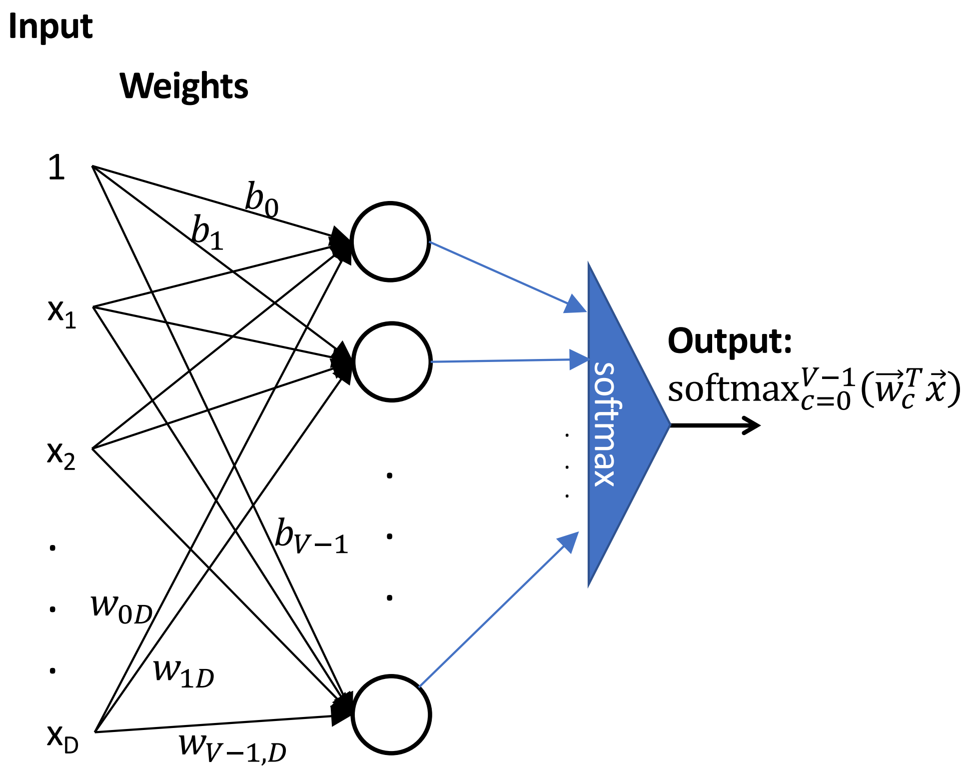
- Each **activation** is a scalar nonlinearity applied to the excitation:

$$h_{ki}^{(l)} = g\left(\beta_{ki}^{(l)}\right)$$

...where $g(\cdot)$ is called the "activation function;" it needs to be chosen in advance by the network designer.



Softmax



Training target is a one-hot vector:

$$\vec{y} = [0, \dots, 0,1,0, \dots, 0]$$

Classifier output is

$$\vec{y}^* = [y_0^*, \dots, y_{V-1}^*], \text{ where } y_c^* = \frac{\exp(\vec{w}_c^T \vec{x})}{\sum_{j=0}^{V-1} \exp(\vec{w}_j^T \vec{x})}$$

We usually train using the cross-entropy loss, also known as negative log-likelihood:

$$\mathrm{L} = -\frac{1}{n}\sum_{i=1}^{n}\sum_{c=0}^{V-1} y_c \ln y_c^*$$

The derivative of L w.r.t. W has a surprisingly simple form:

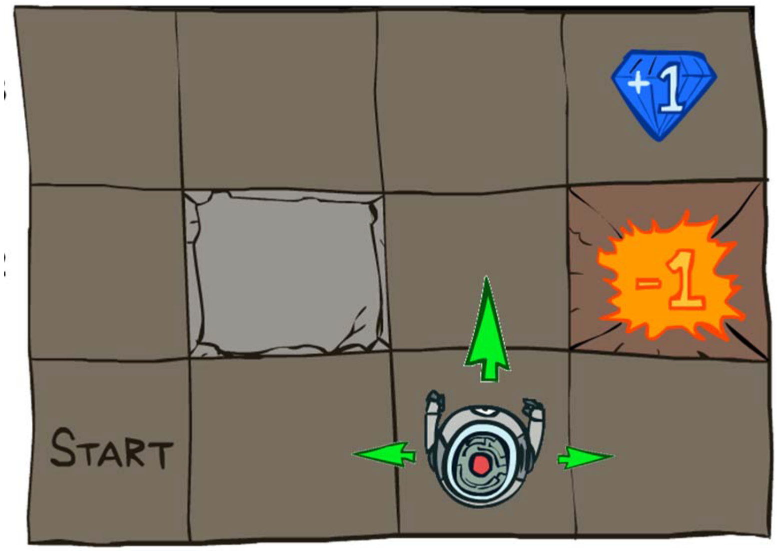
$$\frac{dL}{dw_{cd}} = -\frac{1}{n}\sum_{i=1}^{n}(y_c - y_c^*)x_d$$

Markov Decision Process

- MDP defined by states, actions, transition model, reward function
- The "solution" to an MDP is the policy: what do you do when you're in any given state
- The Bellman equation :

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

- Value iteration:
  - Start with $U^{(0)}(s) = 0$
  - $U^{(t+1)}(s) = R(s) + \gamma \max_a \sum_{s'} P(s'|s,a)U^{(t)}(s')$
- Policy iteration:
  - Start with arbitrary $\pi^{(0)}(s)$
  - $U^{(t)}(s) = R(s) + \gamma \sum_{s'} P\left(s'\middle|s,\pi^{(t)}(s)\right)U^{(t)}(s')$
  - $\pi^{(t+1)}(s) = \operatorname{argmax}_a \sum_{s'} P(s'|s,a)U^{(t)}(s')$



Grid World drawings © Peter Abbeel and Dan Klein, UC Berkeley CS 188

Model-based reinforcement learning

- Start with an initial policy that includes some randomness, governed by an "exploration vs. exploitation" tradeoff, e.g., epsilon-greedy or epsilon-first
- Test a few actions, and **observe** the results
- Based on those results, estimate a **model**: a lookup table (or neural network estimate) of the transition probabilities $P(s'|s,a)$, and of the reward function $R(s)$.
- Based on the model, use value iteration or policy iteration to update your **policy**.
- … and repeat this loop, as often as you can.

Model-free reinforcement learning, e.g., Q-learning

Putting it all together, here's the whole TD learning algorithm:

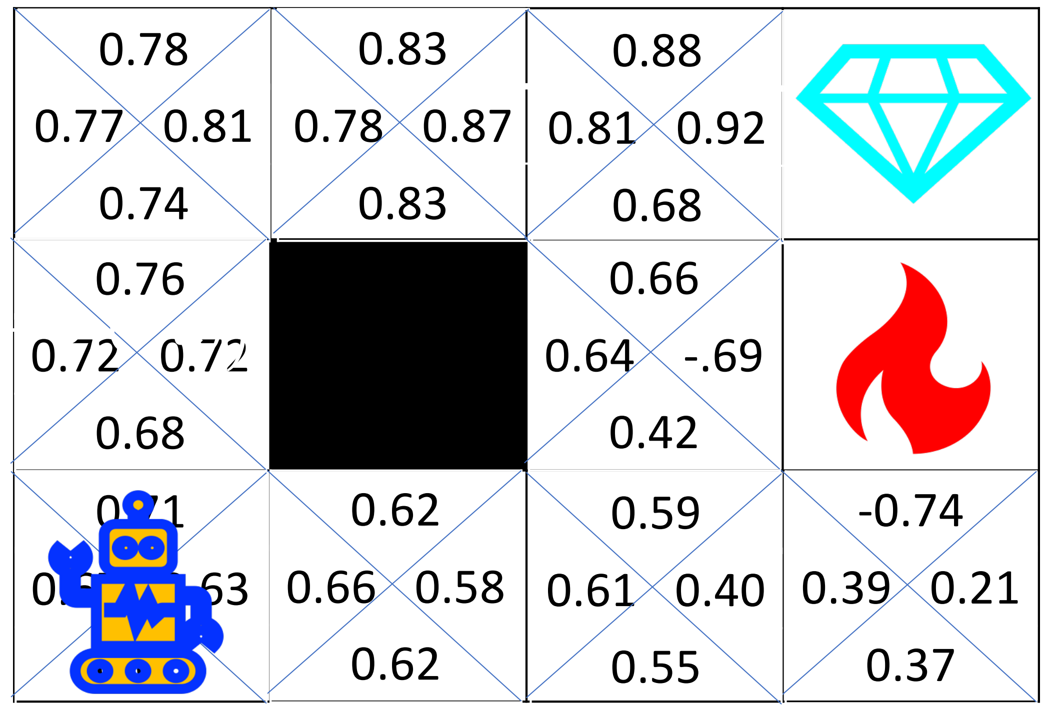
1. When you reach state s, use your current exploration versus exploitation policy, $\pi_t(s)$, to choose some action $a = \pi_t(s)$.
2. Observe the state s' that you end up in, and the reward you receive, and then calculate Qlocal:

$$Q_{local}(s,a) = R_t(s) + \gamma \max_{a' \in A(s')} Q_t(s',a')$$

3. Calculate the time difference, and update:

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha\big(Q_{local}(s,a) - Q_t(s,a)\big)$$

Repeat.



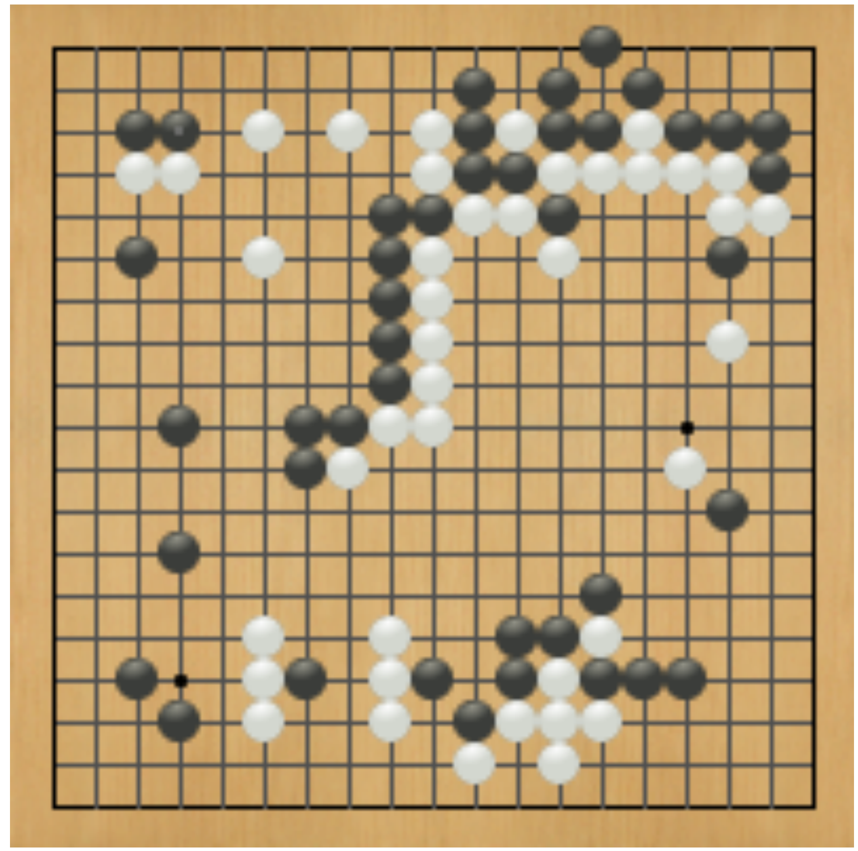
Deep Q-learning and Actor-Critic Learning

1. What is deep Q-learning?
2. How to make Q-learning converge to the best answer?
3. How to make it converge more smoothly?
4. What are policy learning and actor-critic networks?
5. What is imitation learning?

1. Estimate Q(s,a) using a neural net, with Qlocal as training signals.
2. Epsilon-greedy usually works.
3. Experience replay.
4. Actor network: $\Pr(a \text{ is the best action})$. Critic network: $Q(s,a)$, used only to train the actor.
5. Learn to imitate an expert player.

RL for two-player games

- Review: minimax and alpha-beta
  - Complexity: $(2b-1)^{d/2} = O\{b^{d/2}\}$ with depth d and branching factor b, if the children of each node are ordered just right (MAX: largest first, MIN: smallest first)
- Move ordering: policy network
  - Can be used to order the children, with no loss of accuracy; Can also limit the set of moves evaluated, with some loss of accuracy
- Evaluation function: value network
  - Estimates the value of each board position in limited-horizon search
- Exact value: endgames
  - Minimax search backward from a set of known terminal positions
- Stochastic training: Monte Carlo tree search
  - Choose a policy that includes exploration vs. exploitation, play games at random, use the data to estimate win frequency

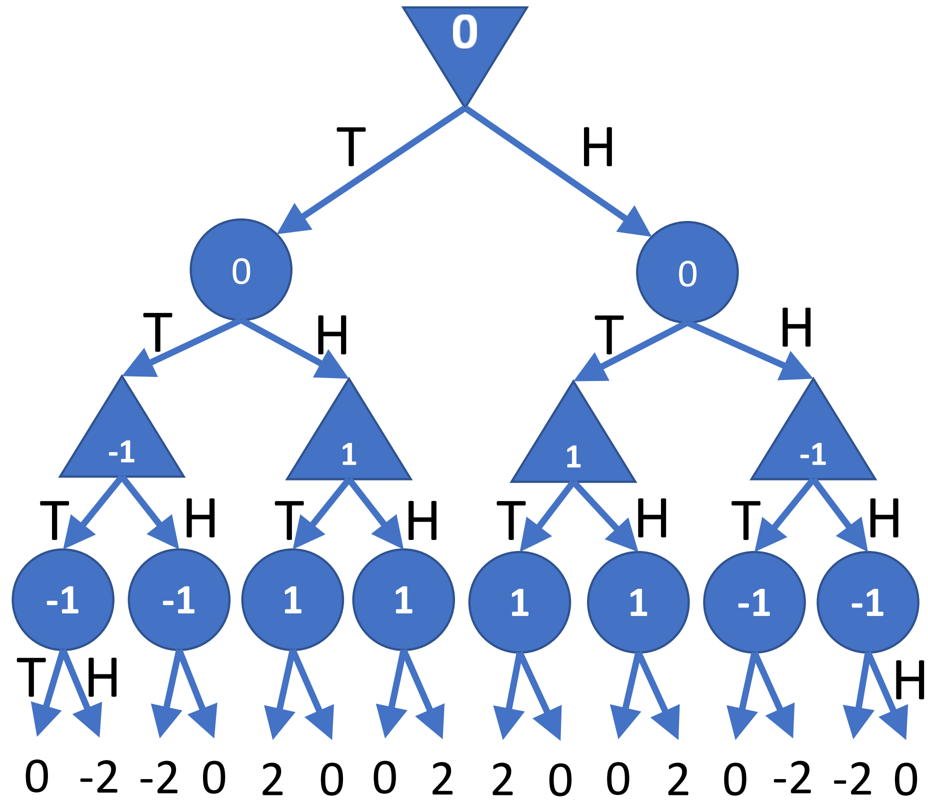


Games of chance and imperfect information

Stochastic games: Expectiminimax

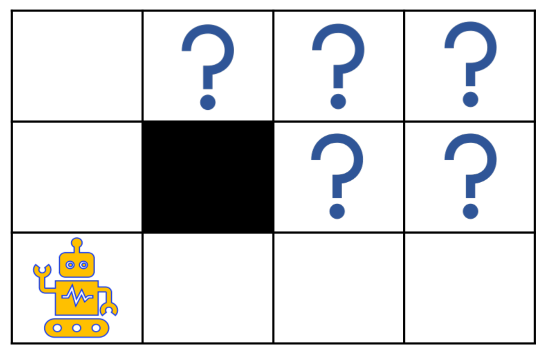
$$U(s) = \max_{a} \sum_{s'} P(s'|s,a)U(s')$$

$$U(s') = \min_{a'} \sum_{s''} P(s''|s',a')U(s'')$$

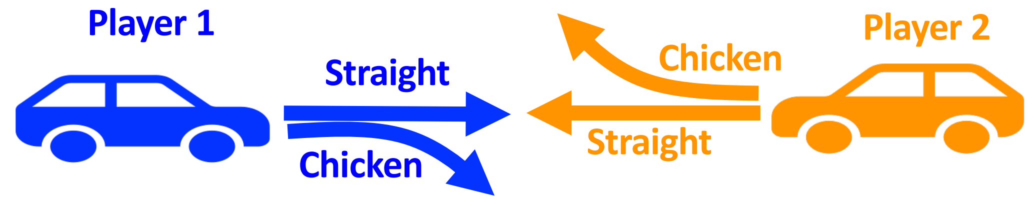


Imperfect information: belief states

$$PREDICT(b,a) = \{s' : s \in b, s' = RESULT(s,a)\}$$
$$UPDATE(b,o) = \{s : s \in b, o = PERCEPT(s)\}$$



Game Theory



- Dominant strategy
  - a strategy that's optimal for one player, regardless of what the other player does
  - Not all games have dominant strategies
- Nash equilibrium
  - an outcome (one action by each player) such that, knowing the other player's action, each player has no reason to change their own action
  - Every game with a finite set of actions has at least one Nash equilibrium, though it might be a mixed-strategy equilibrium.
- Pareto optimal
  - an outcome such that neither player would be able to win more without simultaneously forcing the other player to lose more
  - Every game has at least one Pareto optimal outcome. Usually there are many, representing different tradeoffs between the two players.
- Mixed strategies
  - A mixed strategy is optimal only if there's no reason to prefer one action over the other, i.e., if $0 \le p \le 1$ and $0 \le q \le 1$ such that:

$$(1-p)w + px = (1-p)y + pz$$
$$(1-q)a + qc = (1-q)b + qd$$

	Straight	Chicken
Straight	-10 / -10	1 / -1
Chicken	-1 / 1	0 / 0

	$1-p$	p
$1-q$	w / a	x / b
q	y / c	z / d

Mechanism Design

- Nash equilibrium occurs if:
  - All players have sufficient computation
  - All available actions listed in the payoff matrix
  - Payoff matrix lists true outcome values
- Iterated games:
  - Fixed # games: start from the end, plan backward
  - Random # games: maximize expected gain
- Nash Equilibrium in various auctions:
  - English auction: $b_i = d + \max_{j \neq i} v_j$ iff $v_i > \max_{j \neq i} v_j$
  - Sealed Bid: $b_i = d + \max_{j \neq i} p_{ij}$ iff $v_i > \max_{j \neq i} p_{ij}$
  - Second-Price: $b_i = v_i$, all players
- VCG mechanism to avoid tragedy of the commons: each unsuccessful bidder pays nothing; each successful bidder pays b_{N+1}.

Some sample problems

- Linear classifiers: practice exam problem 4 (today)
- DNNs: next lecture, probably I’ll write a new practice problem
- Q-learning: next lecture, probably I’ll write a new practice problem
- Games of chance & imperfect information: next lecture
- Game theory & Mechanism design: next lecture

Practice Exam Problem 4

You are a Hollywood producer. You have a script in your hand, and you want to make a movie. Before starting, however, you want to predict if the movie you want to make will rake in huge profits, or utterly fail at the box office. You hire two critics A and B to read the script and rate it on a scale of 1 to 5 (assume only integer scores). Each critic reads it independently and announces their verdict. Of course, the critics might be biased and/or not perfect, therefore you may not be able to simply average their scores. Instead, you decide to use a perceptron toclassify your data. There are three features: a constant bias, and the two reviewer scores. Thus f0 = 1 (a constant bias), f1 = score given by reviewer A, and f2 = score given by reviewer B.

Movie Name	A	B	Profit?
Pellet Power	1	1	No
Ghosts!	3	2	Yes
Pac is bac	4	5	No
Not a pizza	3	4	Yes
Endless Maze	2	3	Yes

Practice Exam Problem 4

(a) Train the perceptron to generate Y* = 1 if the movie returns a profit, Y* = -1 otherwise. The initial weights are w0 = -1,w1 = 0,w2 = 0. Present each row of the table as a training token and update the perceptron weights before moving on to the next row. Use a learning rate of eta=1. After each of the training examples has been presented once (one epoch), what are the weights?

Movie Name	A	B	Profit?
Pellet Power	1	1	No
Ghosts!	3	2	Yes
Pac is bac	4	5	No
Not a pizza	3	4	Yes
Endless Maze	2	3	Yes

Practice Exam Problem 4

Iteration	Weights	$y^* = \text{sgn}(\vec{w}^T \vec{x})$	Change weights?	$\eta y_i \vec{x}_i$
1	$[-1,0,0]$	-1	No	
2	$[-1,0,0]$	-1	Yes	$[1,3,2]$
3	$[0,3,2]$	$+1$	Yes	$[-1,-4,-5]$
4	$[-1,-1,-3]$	-1	Yes	$[1,3,4]$
5	$[0,2,1]$	$+1$	No	

Movie Name	A	B	Profit?
Pellet Power	1	1	No
Ghosts!	3	2	Yes
Pac is bac	4	5	No
Not a pizza	3	4	Yes
Endless Maze	2	3	Yes

Practice Exam Problem 4

(b) Suppose that, instead of learning whether the movie is profitable, you want to learn a perceptron that will always output Y* = +1 when the total of the two reviewer scores is more than 8, and Y* = -1 otherwise. Is this possible? If so, what are the weights w0, w1, and w2 that will make this possible?

Movie Name	A	B	Profit?
Pellet Power	1	1	No
Ghosts!	3	2	Yes
Pac is bac	4	5	No
Not a pizza	3	4	Yes
Endless Maze	2	3	Yes

Practice Exam Problem 4

(b) Suppose that, instead of learning whether the movie is profitable, you want to learn a perceptron that will always output

$$Y^* = sgn(A + B - 8)$$

(Y* = +1 when the total of the two reviewer scores is more than 8, and Y* = -1 otherwise). Is this possible?

Answer: yes, sure. For example,

$$w_0 = -8, w_1 = 1, w_2 = 1$$

Movie Name	A	B	Profit?
Pellet Power	1	1	No
Ghosts!	3	2	Yes
Pac is bac	4	5	No
Not a pizza	3	4	Yes
Endless Maze	2	3	Yes

Practice Exam Problem 4

(c) Instead of either part (a) or part (b), suppose you want to learn a perceptron that will always output Y*= +1 when the two reviewers agree (when their scores are exactly the same), and will output Y*= -1 otherwise. Is this possible? If so, what are the weights w0, w1 and w2 that will make this possible?

Movie Name	A	B	Profit?
Pellet Power	1	1	No
Ghosts!	3	2	Yes
Pac is bac	4	5	No
Not a pizza	3	4	Yes
Endless Maze	2	3	Yes

Practice Exam Problem 4

(c)

$$Y^* = \begin{cases} 1 & A - B = 0 \\ -1 & \text{otherwise} \end{cases}$$

Is this possible? If so, what are the weights w0, w1 and w2 that will make this possible?

Answer: NO. This is basically the XOR problem (but in reverse). There is no linear classifier that can solve this problem.