CS188 Intro. to AI
Fall, 2000 R. Wilensky

Final Examination

- This is an open-book, open-notes exam.
- Write your name, etc., in the space below; answer all questions in the space provided. (Space is provided for your name on the top of each page as well.)
- You have 3 hours to work on the exam.
- There are 125 points total.
- Questions vary in difficulty; **do what you know first**.
- Good luck!

NAME: ____________________________

SID: ____________________________ TA: ____________________________

(*Space below for official use only.*)

Problem 1:	__________	(20)
Problem 2:	__________	(25)
Problem 3:	__________	(20)
Problem 4:	__________	(30)
Problem 5:	__________	(20)
Problem 6:	__________	(10)
Total:	__________	(125)

Problem 1

(20 points, 2 points each) For each statement below, say if it is true or false, and give a one sentence explanation of your answer.

(a) The sentence "$\forall\, x,y\ Parent(x,y) \rightarrow Child(y,x)$" is satisfiable but not logically valid.

True: True if Parent and Child are mapped to inverse relations, which of course, they may not be.

(b) Any linearly separable data set can be learned by some single layer perceptron.

True: We proved that perceptrons learn exactly the linearly separable sets.

(c) Decision tree learning algorithms may be subject to overtraining, but not neural network learning algorithms.

False: We meant *overfitting*, which can (and does) occur in NNs as well. (The typo didn't seem to bother many of you, and overtraining is not a bad term for what happens anyway.)

(d) Given the expression (which we corrected during the exam, to include r as an existential variable to and make the ! a 1)

$\exists\, g,a,r,p,d\ \ Ind(g,Giving) \wedge Agent(a,g) \wedge Recipient(r,g) \wedge Donor(d,g) \wedge Theme(p,g)$

we can derive the following expression, assuming that all the constants do not otherwise appear in any other formula:

$Ind(G1,Giving) \wedge Agent(A1,G1) \wedge Recipient\ (R1,G1) \wedge Donor\ (D1,G1\ ) \wedge Theme(P1,G1)$

True: Via Existential instantiation (Skolemization)

(e) Temporal difference learning can be used for deterministic MDPs, but not for non-deterministic tasks.

False: TD-Gammon is a case in point.

(f) If a heuristic always returns the same value no matter what the state, it cannot be admissible.

False. It can still be admissible, just not terribly useful.

(g) The Markov assumption enables the Viterbi algorithm to be computationally tractable.

True. It allows us to limit the amount of state we need to keep track of.

(h) A set of propositions in a "production system", interpreted using a set of conflict-resolution strategies, has the same semantics as they would if interpreted as a knowledge base of logic formulas.

False. In logic, a sentence like "A x P(x)" means that P is true for all x; in a production system, it might mean that P is true for all x except where there is more specific information.

(i) Back-propagation is equivalent to using gradient descent to find a local minimum of an error function over training data.

True. We derived back-prop as such in class.

(j) If we know the joint distribution of two random variables, we can determine the conditional probability of one variable given another.

True. With the joint, we know everything—we can sum up entries to find the individual distributions, etc.

Problem 2 (25 points)

Suppose the following are (two-valued) random variables: **BootsUp**, **OS-Ok**, **HardwareOk**, **SpilledCoffee**. These express the probability that a certain computer will boot (i.e., start up correctly), that its operating system has not been corrupted, that the hardware is functional, and that coffee was spilled on one of the circuit boards, respectively.

(a) (3) Draw a suitable belief network for these variables.

BootsUp has OS-Ok and HardwareOk as parents; HardwareOk has SpilledCoffee as a parent.

(b) (3) Write down the formula that your network expresses for the joint probability distribution of these variables.

P(BootsUp, OS-Ok, HardwareOk, SpilledCoffee) =
P(OScorrupted) ´ P(SpilledCoffee) ´ P(HardwareOk|SpilledCoffee)
´ P(BootsUp|HardwareOk,OScorrupted)

(c) (3) Provide plausible conditional probability tables for the nodes **HardwareOk** and **BootsUp** in your network.

SpilledCoffee	**P(HardwareOk|SpilledCoffee)**
T	**.2**
F	**.95**

HardwareOk	OS-Ok	P(BootsUp |HardwareOk,OS-Ok)
T	**T**	**.99**
T	**F**	**.2**
F	**T**	**.3**
F	**F**	**.06**

(d) (4)

(i) How many *independent* probability values are needed to fill out all the tables required by your network? (Explain your answer in a sentence.)

There are 8 values needed for the probability tables (the other 8 being computable from subtracting from 1).

(ii) How many *independent* probability values would be needed to express the joint if we couldn't make any of the independence assumptions implied by your network structure? (Explain your answer in a sentence.)

We'd have to know 15 values, the last being computable by subtracting the sum from 1.

(e) (2) The conditional probability table for **BootsUp** is a canonical one. Describe, in English, its general structure.

It is a noise-AND. (see above)

(f) (3) Is spilling coffee conditionally independent of whether the OS is corrupted, given that we know whether or not the machine could boot? Can observing the value of another variable change this relationship?

No, but it will be if we observe the remaining node, HardwareOk.

(g) (7) A lie detector test is known to be 99% reliable when the person is guilty but only 90% reliable when the person is innocent. If a suspect is chosen from a group of suspects of whom only 1% have committed a crime, and the test indicates that the suspect is guilty, what is the probability that the suspect is innocent?

P(Innocent | Test=Guilty) = P(Innocent)P(Test=Guilty | Innocent)/P(Test=Guilty)

P(Innocent) = .99
P(Guilty) = .01
P(Test=Guilty | Innocent) = 1-.90=.1
P(Test=Guilty | Guilty) = .99

P(Innocent)P(Test=Guilty | Innocent) = .99 ´ .1 = .099
P(Guilty)P(Test=Guilty | Guilty) = .01 ´ .99 = .0099

by normalization, P(Innocent)P(Test=Guilty|Innocent)/P(Test=Guilty) = .099/(.099+.0099) = .909

Problem 3 (20 points, 2 points each)

Answer true or false, giving a *brief* (one sentence) explanation of your answer.

i) Given a 2-D image, there will generally only be one physically possible 3-D scene that could have created it.

False. There are an infinite number of 2-D scenes that can vary as discussed at length in class and in the text.

ii) Texture can be exploited to segment images and also to help determine the pose of an object.

True. "Texture gradients" help with orientation, etc., texture per se with segmentation.

iii) Stereo disparity is *relatively* more discriminating for objects that are close up than for those that are far away.

True. The angular disparity is $-b\,\delta Z/Z^2$, so even if the distance between the two points changes with the distance for the viewer, the disparity will be linearly smaller.

iv) A primal sketch can be computed by processing all areas of the image uniformly.

True. Uniformly processing the image is an assumption for "early vision" generally.

v) After convolving an image with the derivative of a Gaussian, a "zero-crossing" will indicate the presence of an edgelet.

False: Need to look for a big change in the derivative; zero-crossing in the *second* derivative shows an edgelt.

vi) We can assign unique labels to lines in a scene only if the scene consists of instances of objects for which we have models.

False. An object model is a model of a particular kind of object. We don't make any assumptions about particular kinds of objects for labeling, just assumptions about the nature of their geometry, etc.

vii) A catalog of legal junction types can be used to constrain possible line labels, but the catalog will get larger as we enlarge the class of scenes we wish to interpret.

True. As we include cracks, shadows, etc., the catalog can get large.

viii) Generalized cylinders are proposed to describe the shape of objects because it is not possible to approximate some object shapes using polyhedra.

False. It is always possible, just sometimes costly.

ix) The shading of an object provides useful information about that object's shape.

True. This is "shape from shading".

x) Color does not appear to play a significant role in computer vision because it provides no useful information for segmentation.

False. It is quite useful, as we saw in Blobworld, just hard to do right.

Problem 4 (30 points)

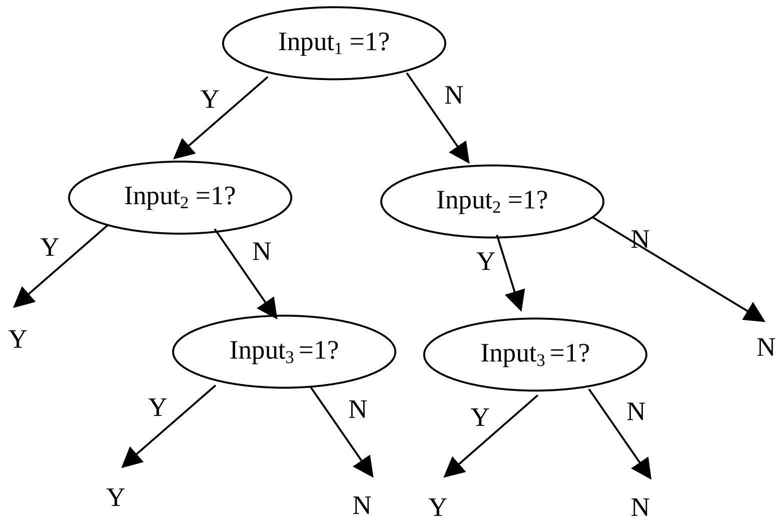
(a) (5) Let's define the "majority" function of *n* binary inputs $x_1, \ldots, x_n$, each either -1 or 1, to be 1 if more than half of the inputs are 1, and -1 otherwise. Can this function be represented by a single layer perceptron? Either show that this is impossible or construct such a perceptron (i.e., for a given *n*, specify what the weights for such a perceptron could be).

Sure, set all the weights to 1/n and the threshold to ½, say.

(b) (5) The function EQUAL of two inputs, x_1 and x_2, is defined to be 1 if the inputs are the same (both -1 or both 1) and -1 otherwise. Can this function be represented by a single layer perceptron? Either show that this is impossible or construct such a perceptron.

This is just ``not XOR''; XOR isn't linearly separable.

(c) (5) Draw a *decision tree* for the majority function, assuming three inputs.

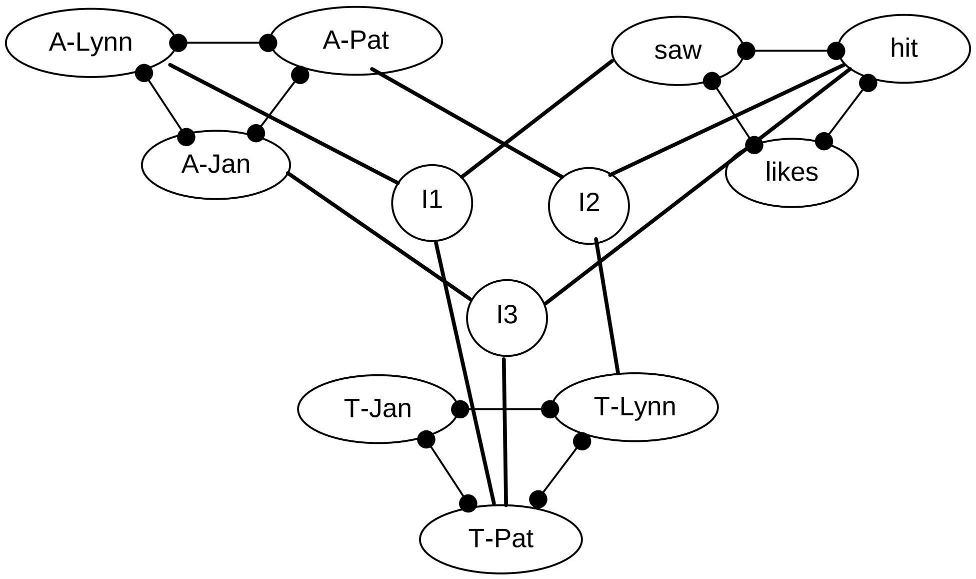


(d) (5) We observed (although we didn't prove) that any function can be approximated by a multilayer feedfordward neural network of a few layers in which the hidden units have sigmoid activation functions. Suppose instead we used linear units, i.e., units whose activation function was just a constant multiple of its total input. Doing so would severely restrict the expressive power of the networks that such networks could implement. Explain why.

If all the nodes were linear, the whole network could only compute a linear combination of its inputs, which is a quite restricted set of functions.

Below is a network of units. The units are represented by large and small ovals, all functionally identical. The large ovals are intended to represent people in different roles, and action types, and the small ones, actions. The light lines with dots at each end represent *inhibitory* connections between units, and the dark lines *excitatory* connections; all connections are bidirectional.

For example, the oval labeled “A-Jan” represents Jan in the role of an agent, and the one labelled “A-Lynn” Lynn being an agent. These are shown as inhibiting one another. However, the node labeled “A-Lynn” is connected by an excitatory link to a small circle labeled “I1”, which is itself connected to an oval labeled “saw” (i.e., the action of seeing) and to one labeled “T-Pat” (Pat being the theme of some action, i.e., the thing acted upon). In other word, the unit I1 represents an event in which Lynn saw Pat.



Assume the all units are at a comparable initial activation. Now suppose we “clamp down” (i.e., activate continually) the units labelled “hit” and “T-Pat” and propagate activations throughout the network.

(e) (3) What other units should end up with high activations? In general, how would you characterize the computation that this network is performing? (I.e., what more traditional computer science system is this network approximating?)

I3 and A-Jan would have high activations. In effect, the network is running a query, or implementing associative memory. I.e., given some properties, it will activate entities with those properties, along with other properties of those entities.

(f) (2) If we activate only the unit labelled “A-Pat”, the unit for T-Pat will go up in activation somewhat (although, perhaps, not as much as the one for T-Lynn). Since Pat doesn’t do anything to herself in any of these actions, why does this unit's activation move up at all?

A-Pat activates I2, which activates “hit”, which activates I3, which activates T-Pat. In effect, we are activating units associated with the associations of our initial unit. This is in effect a kind of

"crosstalk". In general, this is a real problem in NNs: It is very hard to create the equivalent of variable bindings in NNs.

(g) (5) In reinforcement learning, algorithms like Q Learning tell us how to successively approximate some function, which, if we knew it, would in turn give us an optimal policy. We have to explore our world in order to estimate this function, and to do so, we have to move around via *some* policy. Indeed, we may already have learned quite a decent policy, and it would seem reasonable to use the best policy we know of to wander round, as we try to learn a better policy.

However, rather than use the best policy estimate directly, many reinforcement learning algorithms instead do the following: Most of the time, follow the best estimate policy; some small percentage of the time, *pick one of the possible actions at random*.

Picking an action at random is likely to give an inferior result for that particular action, especially when our policy is already pretty good. Why is doing so nevertheless a good idea?

Many reinforcement learning algorithms, e.g., Q Learning, require that we visit each state infinitely in order to be sure they converge on the optimal policy. Picking an action at random, even a small percentage of the time, assures this is the case.

Problem 5 (20 points)

(a) (5) Each of the following examples illustrates a kind of natural language ambiguity. In each case, name a type of ambiguity involved, explaining the particular instance of ambiguity in the sentence that must be resolved. Be as specific as possible; e.g., state whether a lexical ambiguity is syntactic or semantic (or both).

i. "Squad helps dog bite victim" (actual newspaper headline)

Syntactic ambiguity: [np squad] [v helps] [np dog bite victim] or
[np squad] [v helps] [np dog] [vp [v bite] [n victim]]]

(also, bite is lexically syntactically ambiguous (n in first interpretaion, v in second)

ii. "I had a ball."

semantic lexical ambiguity of "ball"

iii. “John told the waiter he didn’t have any money.”

pronoun reference (“he” might be the waiter or John)

iv. “The judgment of the court has been questioned.”

role ambiguity: The court can be the agent of the judgment, or its theme (if, e.g., the public no longer trusts the court).

v. “Jan doesn’t want to marry someone”.

scope of quantifier ambiguity: It’s not the case that there exists someone that Jan wants to marry, versus, there exists some whom Jan doesn’t want to marry.

$\neg\ \exists p$ Want(Marry(Jan, p))

$\exists p\ \neg$ Want(Marry(Jan, p))

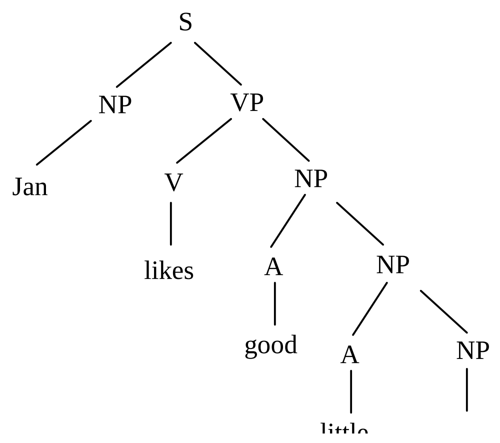
(b) (5) Consider the following context-free grammar:

S → NP VP
VP → V NP
NP → D NP
NP → A NP
NP → N
NP → Jan
D → the, a
A → little, young, good
N → boy, girl, boys, girls
V → likes, like

Indicate (by circling Y or N) which of the following sentences are admitted by this grammar:

i.	Jan likes a little boy.	**Y**
ii.	Likes a young boy a little girl.	**N**
iii.	Boys like girls.	**Y**
iv.	Little a young boy likes the a girls.	**Y**
v.	A Jan likes a boy.	**Y**

(c) (5) Show a parse tree for the sentence “Jan likes good little girls.”



Name, SID: ____

(d) (5) We can characterize a problem with our grammar as follows: Some nouns or noun phrases are ready to be used elsewhere, and others are not, but our grammar doesn't make this distinction. For example, "boy" and "boys" are both described as Ns by a single rule, but "boys" can be a noun phrase all by itself (as in "boys will be boys"), while "boy" requires "finishing" (i.e., "a boy will be a man" but not "boy will be man").

Suggest two strategies that a grammar-designer might use to address this problem, one that stays within the confines of the context free grammars, and another than does not. (You need not provided a detailed solution. Just tell us how to approach the problem.)

Using CFGs, one can create lots more non-terminals and rules. E.g., instead of N, one can have $N_{unfinished}$ (for "boy", say) and a $N_{finished}$ (for "Jan", say) and a $N_{neither}$ (for "boys", say), and then add lots of rules to deal with these (creating $NP_{finished}$, etc., also).

Alternatively, one could use unification grammars, and try to introduce features to mark Ns for these properties, passing them along and constraining them appropriately.

Problem 6 (10 points)

(a) (5) The following is a successor-state axiom for a "Go" operator:

$$\begin{aligned}\forall x,y,z,s,a\ \ At(y,Do(a,s)) \leftrightarrow\ & (At(x,s) \wedge a=Go(x,y)) \\ & \vee (At(y,s) \wedge \neg (a=Go(y,z) \wedge \neg\, y=z))\end{aligned}$$

Give a STRIPS-style representation for the same operator. What information is explicit in the successor-state axiom that is implicit in the STRIPS-style operator?

Go(x,y):
pre: At(x)
del: At(x)
add: At(y)

The fact that not moving leaves things where they are is not expressed.

(b) (5) Suppose it were true that *At(Loc1,S_1)*. We would like to use resolution to prove from this fact and the successor-state axiom above that we can move to *Loc2*. What difficulty would we have in using resolution given the form of this axiom?

Fortunately, we don't need all the information in this axiom to prove what we want, but just that part that says "If we are at a location in a state, then moving to a location results in a state in which we are at that location". Write down this part of the above axiom in clausal form.

Equality makes resolution inapplicable. The necessary part is just

$\forall$ **x,y,s At(x,s)** $\rightarrow$ **(At(y, Do(Go(x,y),s))**

which, in clausal form is

{$\neg$ At(x,s), At(y,Do(Go(x,y),s))}