2nd Midterm Exam

CSC 242

6 May 2005

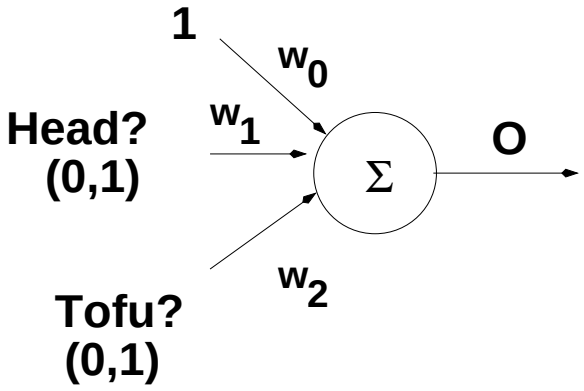
*Write your **NAME** legibly on the bluebook. Work all problems. You may use three double-sided pages of notes. Please hand your notes in with your bluebook. The best strategy is not to spend more than the indicated time on any question (minutes = points, total of 120).*

FFQ: This is the second test this year I cribbed from past tests w/o saying anything. Some people figured it out, others did not seem to. Generally the knowledge was helpful I think, but sometimes not entirely so (see Q2 below). Also we won't be cribbing from past tests again for a couple of years.

1. Perceptron Learning: 25 Min

FFQ: My sign error in this problem made it much harder than I had intended. Below is what I SHOULD have said. As it was, 15 points to everyone, 20 if you showed you knew perceptron learning rule, 25 if you noticed that only learned with mistakes.

You want your Quagent to learn the AND function, which is true only if its inputs are both true. In particular, you want to train your Quagent to hold both the Head and Tofu. Your perceptron looks like this:



The input I_0, set to a constant 1 with weight w_0, implements the threshold $t = w_0$. I_1 is 1 if the Head is held and 0 if not. The Tofu input I_2 is similar. Your initial weights are $(w_0, w_1, w_2) = (-2, 1, 0)$.

A (10 Min). **Either draw the resulting decision surface or evaluate O, the output of the perceptron, for the four possible input values (1,0,0), (1,0,1), (1,1,0), (1,1,1)**.

B (5 Min). Part A should have demonstrated that your quagent has some learning to do. Thought experiment: suppose you use the perceptron learning rule (any learning rate) with inputs (1,0,0), (1,1,0), (1,0,1)? **What happens to the weights?**.

C (10 Min). Now actually use the perceptron learning rule with input (1,1,1). Say you want a positive output ($O > 0$) if the AND is true, else a negative output ($O < 0$): you let the error E be 1 if the output is the wrong sign for that input and 0 if it is the right sign. Use learning rate $\alpha = .4$.

What are the weights after one training trial and how well does the perceptron perform now? *I.e.*, **either draw the resulting decision surface or evaluate O, the output of the perceptron, for the four possible input values (1,0,0), (1,0,1), (1,1,0), (1,1,1)**.

Answer:

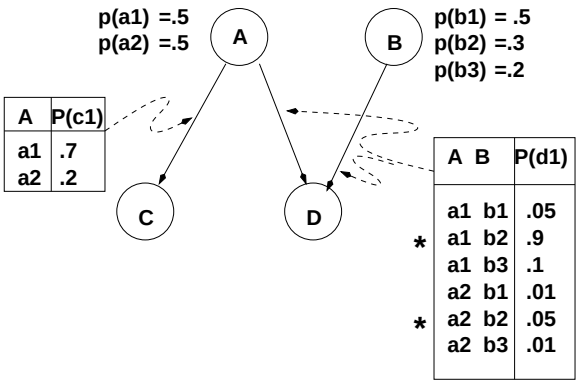
A. The decision surface is a vertical line off to the right of all four inputs.

B. Three of the inputs (in fact, those three) give correct outputs, so (since perceptron learning is driven by error), training with them will not change any weights.

C. Rats, I lost my notes. Anyway as I recall after one trial with this learning rate, the perceptron converges to the right answer with some set of weights I misremember. The decision line tilts over to the left and drops enough to exclude the point (1,1) and include the other three points.

2. Bayes Nets: 20 Min

FFQ: Generally people either totally screwed this up or copied the answer off their Xeroxes of the previous exam's answer. Only slight problem there was an obvious misprint which, if perpetuated, lost 5 points. In the end, I think maybe only one person got the right answer.



Shown is a Bayes Net relating four random variables. Variable B can have three values, the others are binary (values of "true" (a_1, c_1, d_1) or "false" (a_2, c_2, d_2). Clearly $P(a_2) = (1 - P(a_1))$, etc. Shown are the prior probabilities of A and B and the conditional probability matrices (CPMs) linking A, B, C, D.

A (15 Min). Suppose you find out that C is true (i.e. C's value is c_1). Ignoring the numerical values involved, **what is the resulting probability of b_2?** That is, we need a nice neat formula into which we can substitute things we know.

B (5 Min). Now considering numbers – suppose the entries for $P(d_1)$ in the two starred rows in the CPM are swapped. **Does the answer to part A go up or down and why? Do you actually have to answer part A to figure this out and if not how not?**

Answer:

A.

Lots of people thought that the "causal" arrows meant that variables were independent, but Bayes' rule says that isn't the case, right? I expected people just to write down a version of the formula for a similar example from the book but most people who got this did it by brute force, figuring out how to rewrite the probabilities by applying Bayes, working around the tree.

$$P(b_2 \mid c_1) = \alpha \sum_A \sum_D P(b_2)P(A)P(D \mid A, B)P(c_1 \mid A)$$

You can move non-summed-over factors outside various sums to get

$$\alpha P(b_2) \sum_A P(A) \sum_D P(D \mid A, B) P(c_1 \mid A)$$

which looks to take just four terms of 4 factors apiece to calculate.

B. Most people got this right.

You could substitute into your sums if you wanted and show what happens.

In general you don't need the formula though. You could explain about Bayes theorem and even quote it, but the intuition is that C makes A more likely, and A and b2 make D more likely with the original CPM. Swapping the two probabilities means that it is much less likely that d_1 occurs with b_2, so the answer should go down.

3. Linear Systems: 25 Min

A (5 Min). Attached figure "Images and Spectra" has four images and four power spectra. **Supply the letter of the corresponding power spectrum for each image's number.**

1:
2:
3:
4:

B (10 Min). Suppose you are designing a digital camera. You plan to have a FFT chip that furnishes the power spectrum (in real time, of course) for your internal use. Also of course you'll have the image that gets passed to the consumer. **How would you implement an** *autofocus* **capability for the camera given the power spectrum?**

C. (10 Min) **How would you use an image transform like the FFT to do image compression?.**

Answer:

A.

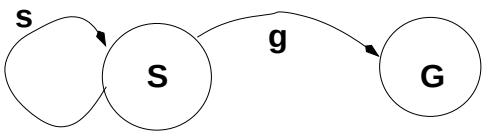
1:D
2:A
3:B
4:C

B. Move focus in and out while monitoring the energy in the high-frequency part of the spectrum by integrating the power spectrum for frequencies above some fraction of the band-limit of the particular image you're getting. Stop at the maximum.

C. I gave half credit for the idea of tossing out low coefficients, and half for the idea you had to do this locally, not on the whole image. Unfortunately nobody got the second 5 points!

Tile the image up into small squares (maybe 8x8 pixels), take the FFT (or if you're JPG, the Discrete Cosine Transform) of the patch and keep the largest coefficients of the frequencies. Keeping them all allows perfect recovery but no compression. In the limit of one tile per image, you apply the same attenuation of wavelengths everywhere, which is not likely to be useful (could vary from edge-enhancement to blurring to something like emphasizing the "midrange", done uniformly all over the image).

4. Reinforcement Learning: 30 Min



R(S) = r = −0.1 **R(G) = R = 1.0**

The above rather simple world has a start state S and a goal state G, with action s (for "stay" or "start") that is meant to keep the agent in the start state, and g (for "go" or "goal") meant to take agent to goal. The world is completely observable to the agent (it knows what state it's in and what reward it's getting). At goal, agent exits the game; there are no more moves until its next trial. Say there is a future-reward discount factor of γ. Reward of the state S is r and the reward of the goal G is R, which in this world are -0.1 and +1.0 respectively. Thus if the agent wakes up in S and takes action g successfully, its total reward is $-0.1 + 1.0$.

The world is uncertain. In fact the transition function $T(FromState, Action, ResultState)$ giving the probabilities of transitions given actions is: $T(S, s, S) = .9, T(S, s, G) = .1, T(S, g, S) = .2, T(S, g, G) = .8$. There aren't any $T(G, ., .)$ since in state G there aren't any moves, the game is over. Call $T(S, g, G) = p$, the probability of successfully "go"ing, and then $T(S, g, S) = q = 1 - p$, the probability of an unsuccessful "go" action.

A (1 Min). **What is the agent's optimal policy?** (Don't get this wrong!)

B (10 Min). Using the normal assumption of discounting future rewards by a factor of γ per timestep, **what is the utility of S in the above situation?** Ideally, I'd like to see your work leading to an explicit, symbolic, closed-form solution. Less ideally, give a coherent approach to such a solution. A final number is not necessary.

C (9 Min). If $\gamma = 1$ (additive rewards), **write down your thus-simplified formula (or make a new one) and give your best estimate of the numerical value of the utility of state** S using your formula (better) or your intuition.

D (10 Min). Now assume the agent knows neither how the world works nor the utilities of any states. However suppose it is following, by luck or accident, the optimal policy. It makes four trials in the world; in each trial it emits the best action until it finally gets to G. The trials yield the following state (not action!) sequences: SSG, SG, SG, SSSG. **What is the agent's resulting estimate of** $T(., ., .)$**? What technique would be appropriate for learning the values of** T **over such a sequence of trials?**.

Answer:

A. Optimal policy is always to "go". This has got to be obvious by inspection: hanging around costs, there's only one other thing to do...

B. Here I pretty much gave full credit to anyone (and there were precious few) who realized there was a nontrivial, in fact infinite, sum involved.

First step, agent does "g", and with probability p gets total reward $r + \gamma R$. If there's a step 2, then with probability pq the reward is $r + \gamma r + \gamma^2 R$. And so on, so I get the sum of these outcomes weighted by their probabilities as:

$$U(S) = \sum_{t=0}^{\infty} pq^t [\gamma^{t+1} R + \sum_{k=0}^{t} \gamma^k r]$$

The second sum simplifies to $r\frac{(\gamma^t - 1)}{(\gamma - 1)}$.

C. With $\gamma = 1$, the above simplifies to

$$U(S) = \sum pq^t[R + rt] = 0.875.$$

This isn't quite as intuitive to me as another formula, which describes your getting R except when you don't, which is the first time (you get 1- .1), and then with probability decreasing by factors of .2 every time, you lose another 0.1:

$$U(S) = 1 - .1 - .2(.1) - (.2)^2(.1)... = 1 - (.1)\sum_{t=0}^{\infty} .2^t = 1 - (0.1)\frac{1}{1 - .2} = 0.875.$$

D. Lots of people got this right.

I make it $T(S, g, S) = 3/7$ and $T(S, g, G) = 4/7$. Clearly no info on $T(S, s, .)$ since agent following best policy would never do that, and none on $T(G, ., .)$ since those are terminal states.

5. Short Answers: 20 Min

Define, discuss, or describe these terms in a sentence or two (2 Min apiece).

1. We've discussed the delta rule for perceptron learning and the back-propagation rule for feed-forward multi-layer net learning. How could we use other search or optimization techniques we've studied to find correct weights in neural nets?

2. Attached figure "Textures" has some sample texture images. How can we use the FFT for texture classification (that is, what useful features can we find in spatial-frequency space?)

3. Looking at "Textures" again, how could we use edge finders classification (that is, what useful features can we build out of edge-finders?)

4. What are some advantages of tops-down control for a vision system?

5. What are some advantages of bottoms-up control for a vision system?

6. What's the Modulation Transfer Function?

7. What's a Low-Pass Filter?

8. What's a Hough Transform?

9. What's "Cubist Vision?"

10. What does the convolution theorem for Fourier transforms tell us and why is it useful?

Answers:

1. Delta and backprop are forms of gradient ascent, which is why you need a differentiable activation function to use them for multilayer perceptrons. Also the rules derive from particular choices of error metric (squared error) at the outputs. One could imagine other error choices and the whole range of nonlinear optimization techniques, such as one-dimension at a time search (change a random weight and if there's an improvement, keep it else try something else), simulated annealing (or Cauchy or Boltzmann training) or genetic algorithms. Hybrid approaches are also possible, with hill-climbing (aka backprop) alternating with some random component to get off of local maxima.

2. Natural features are the amount of energy in concentric annuli or radial pie-slices (or a combination: low-freq. in a direction, say) out of the power spectrum. Power spectrum is of course $F(u,v)F*(u,v)$. One can get rotation invariance by some straightforward manipulation of the results.

3. Here one can pretty much duplicate the power spectrum results by running edge detectors of varying spatial support (blurriness) and making features out of the edge density, histograms of edge direction, etc. at different spatial resolutions.

4. Tops-down control is like "controlled hallucination" in which the assumption is we pretty much know what we're going to be seeing and only need to verify a few details: it's goal-oriented and thus rather like backward chaining. It's easier to verify a visual phenomenon than to extract and describe it.

5. On the other hand, if you're not expecting an important phenomenon then tops-down control may miss it. And you might argue that vision is exactly to deliver "surprises", i.e. information, to the brain. A bottoms-up approach that, say, considers the entire image instead of some favored region may lead to important discoveries.

6. Modulation Transfer Function: Fourier transform of impulse response: gives attenuation and amplification of frequencies by the linear system being described.

7. Low-Pass Filter: throws away high-frequency components of a signal.

   FFQ: Low-PASS filter, not Low-BLOCK filter!!

8. Hough Transform: mode-based estimation strategy based on voting: explanation of data with most votes wins. Deals with outliers better than mean.

9. Cubist Vision: Randal Nelson's four-tier approach to vision using curve segments, contextual patches, and multiple views.

10. The Convolution theorem states that multiplication and convolution are Fourier transfer pairs, so you can multiply elementwise in the Fourier domain to get the FT of the convolution of two functions in the spatial domain. Not only is this a hugely powerful conceptual tool (think of graphic equalizers) but with the FFT you get an $O(n \log n)$ vs. $O(n^2)$ computational advantage.