Final Exam: Artificial Intelligence

2017 Fall

January 8, 2018

1 Short Questions

(a) Recall in uniform-cost search, each node has a path-cost from the initial node (sum of edge costs along the path), and the search expands the least path-cost node first. Consider a search graph with $n > 1$ nodes (n is an even number): $1, 2, \ldots, n$. For all $1 \leq i < j \leq n$ there is a directed edge from node i to node j with an edge cost j. The initial node is 1, and the goal node is n. How many goal-checks will uniform-cost search perform?

Either following answer works

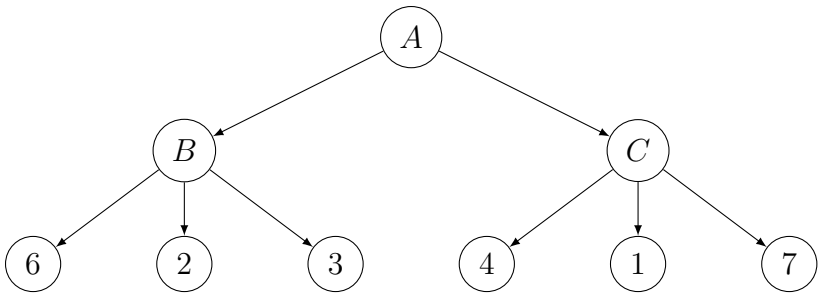
Answer One: n (using graph search)

Answer Two: 2 + the number of paths from node 1 to node$i$ with length less than n for $\forall i, 1 < i < n$ (using tree search) .

(b) Given two admissible heuristic functions h_1, h_2, what range of α guarantees that $h_1 - \alpha(h_1 - h_2)$ is admissible?

$\alpha \in [0, 1]$, this is the convex combination (weighted average) of the two heuristics.

(c) Which nodes are pruned by alpha-beta pruning? The max player moves first.



Node 7 will be pruned

(d) If the only difference between two MDPs is the value of the discount factor then must they have the same optimal policy? Justify your answer.

No. A counterexample suffices to show the statement is false. Consider an MDP with two sink states. Tran- sitioning into sink state A gives a reward of 1, transitioning into sink state B gives a reward of 10. All other transitions have zero rewards. Let A be one step North from the start state. Let B be two steps South from the start state. Assume

actions always succeed. Then if the discount factor ¡ 0.1 the optimal policy takes the agent one step North from the start state into A, if the discount factor ¿ 0.1 the optimal policy takes the agent two steps South from the start state into B.

(e) Consider a classification problem with 10 classes $y \in \{1, 2, \ldots, 10\}$, and two binary features $x_1, x_2 \in \{0, 1\}$. Suppose $p(Y = y) = 1/10, p(x_1 = 1|Y = y) = y/10, p(x_2 = 1|Y = y) = y/540$. Which class will naive Bayes classifier produce on a test item with $(x_1 = 0, x_2 = 1)$?

$$\begin{aligned}
&\text{argmax}_y P(y \mid x_1 = 0, x_2 = 1) && (1)\\
&= \text{argmax}_y P(x_1 = 0, x_2 = 1 \mid y)p(y) && (2)\\
&= \text{argmax}_y P(x_1 = 0 \mid y)P(x_2 = 1 \mid y)p(y) && (3)\\
&= \text{argmax}_y (1 - P(x_1 = 1 \mid y))P(x_2 = 1 \mid y)p(y) && (4)\\
&= \text{argmax}_y (1 - y/10)y/540 && (5)\\
&= \text{argmax}_y (10 - y)y && (6)
\end{aligned}$$

Now you could enumerate y to find the maximum. Or pretend y is continuous, taking derivative and set to zero: 10-2y=0 leads to y=5.

(f) In a neural network, which techniques are used to deal with overfitting? Please suggesting three techniques.

Dropout, regularization, batch normalization, or cross validation

(g) In a neural network, how to deal with underfitting?

increasing the model capacity, e.g., adding more hidden layers or more nodes in hidden layers

(h) A convolutional neural network has 4 consecutive 3x3 convolutional layers with stride 1 and no pooling. How large is the support of (the set of image pixels which activate) a neuron in the 4th non-image layer of this network?

With a stride of 1, and a 3x3 filter, and no pooling, this means the outer ring of the image gets chopped off each time this is applied. Hence, this reduces the dimension from $(n \times n)$ to $((n-2) \times (n-2))$. We get the answer, working backwards. Thus, the support is 9x9=81 pixels.

(i) What is the role of pooling layers in a Convolutional Neural Network?

Pooling layers have several roles: (a) they allow flexibility in the location of certain features in the image, therefore allowing non-rigid or rotated or scaled objects to be recognized, (b) they allow unions of features to be computed, e.g. blue eyes or green eyes (c) they reduce the output image size (or reduce the computational requirements, (d) reducing the likelihood of overfitting.

(j) What reasons do you think of using Rectified Linear Units (ReLu) in deep neural networks? Rectified linear units propagate the gradient efficiently and therefore reduce the likelihood of a vanishinggradient problem that is common in deep neural architectures. Rectified linear units threshold negative values to zero, and therefore solve the cancellation problem as well as result in a much more sparse activation volume at its output. The

sparsity is useful for multiple reasons but mainly provides robustness to small changes in input such as noise(6). Rectified linear units consist of only simple operations in terms of computation (mainly comparisons) and therefore much more efficient to implement in convolutional neural networks.

(k) For the policy gradient update below, what is the role of the baseline?

Initialize policy parameter θ, baseline b
for iteration=$1, 2, \ldots$ **do**
  Collect a set of trajectories by executing the current policy
  At each timestep in each trajectory, compute
   the *return* $R_t = \sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'}$, and
   the *advantage estimate* $\hat{A}_t = R_t - b(s_t)$.
  Re-fit the baseline, by minimizing $\|b(s_t) - R_t\|^2$,
   summed over all trajectories and timesteps.
  Update the policy, using a policy gradient estimate $\hat{g}$
   which is a sum of terms $\nabla_\theta \log \pi(a_t \mid s_t, \theta)\hat{A}_t$
end for

Variance reduction. Rewards (and returns) can be large, vary dramatically over time, and may also always be positive. That means the terms that sum to form a policy gradient have large magnitude. Although they may cancel to produce an unbiased result, the sum has large variance. An advantage estimator instead has approximately zero mean, and is formed by subtracting from the return a baseline estimator computed over the entire state space. The terms in the sum for are now smaller and the sum has lower variance. In the simplest case, the baseline is a linear function of the current state.

2 Search (10 Points)

Consider the following search problem. There is a set of operations $O = \{o_1, \ldots, o_n\}$, and a set of conditions $C = \{C_1, \ldots, C_m\}$. Each operation $o_i \in O$ has a set of preconditions $P_i \subseteq C$, and a set of effects $E_i \subseteq C$. A state is defined by a subset of conditions $S \subseteq C$. An operation $o_i \in O$ can be applied at state S if and only if $P_i \subseteq S$, and it leads to the state $S \cup E_i$. The goal state is C, i.e., the state that contains all conditions. The initial state is the empty set (so initially only operations o_i that have an empty P_i can be applied).

(a) Design a consistent heuristic for this search problem and prove that A* graph search with this heuristic is optimal (it always finds the shortest sequence of operations that leads to the goal state). You may rely on any theorem stated in class. (10 Points)

We define the following heuristic function h for this search problem. Given a state S, $h(S)$ computes the optimal path to the goal state, in the modified problem where every operation o_i is replaced with the operation o_i', which has the same set of effects E_i, but an empty set of preconditions. Informally, any of the "old" operations can be applied at any state. (The perceptive student may have noticed that computing $h(S)$ is equivalent to solving the Minimum

Set Cover problem, that is, computing $h(S)$ happens to be computationally hard, so this is a pretty bad heuristic.)

We know from class that A* graph search with a heuristic h is optimal if h is consistent. We will now prove that h is consistent; i.e., that $h(x) \leq h(y) + c(x, y)$.

Let $H(x, y)$ be the minimum number of moves necessary to get from x to y under the conditions of the heuristic function. Note that $H(x, C) = h(x)$. Because h is a relaxation of the search problem, we know that $H(x, y) \leq c(x, y)$.

If we consider any x and y, we can see that $h(x)$ is at most the number of moves under h to get from x to y plus the number of moves under h to get from y to C; in other words,

$$h(x) \leq H(x, y) + h(y).$$

However, we also know that $H(x, y) \leq c(x, y)$, which means as desired.

$$h(x) \leq c(x, y) + h(y)$$

,

3 Reinforcement Learning (14 Points)

Assume we have an MDP with state space S, action space A, reward function $R(s, a, s')$, and discount γ. Our eventual goal is to learn a policy that can be used by a robot in the real world. However, we only have access to simulation software, not the robot directly. We know that the simulation software is built using the transition model $T_{\text{sim}}(s, a, s')$ which is unfortunately different than the transition model that governs our real robot, $T_{\text{real}}(s, a, s')$.

Without changing the simulation software, we want to use the samples drawn from the simulator to learn Q-values for our real robot.

Recall the Q-learning update rule. Given a sample (s, a, s', r), it performs the following update:

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

(a) Assuming the samples are drawn from the simulator, how can we modify the Q-learning update rule so that it will learn the correct Q-value functions for the real world robot? Provide an explanation why your modification works. Assume $T_{\text{sim}}(s, a, s')$ and $T_{\text{real}}(s, a, s')$ are known. (10 Points)

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha \frac{T_{\text{real}}(s, a, s')}{T_{\text{sim}}(s, a, s')} \left[r + \gamma \max_{a'} Q(s', a') \right]$$

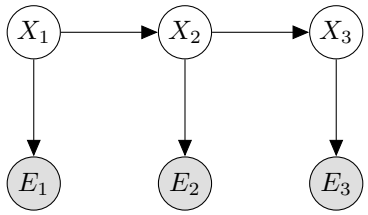
Given a state-action pair (s, a), the simulation will sample a next state according to $T_{\text{sim}}(s, a, s')$. However, that transition actually occurs as frequently as $T_{\text{real}}(s, a, s')$. To account for this bias during sampling, we will adjust the weight of the sample by dividing by the simulation probability. Specifically, each sample should be weighted by $\frac{T_{\text{real}}(s,a,s')}{T_{\text{sim}}(s,a,s')}$. In this way we use weights to adjust the Q-value function updates so that they are correct in expectation instead of sampling from the correct distribution directly.

(b) Now consider the case where we have n real robots with transition models $T^1_{\text{real}}(s, a, s'), \ldots, T^n_{\text{real}}(s, a, s')$ and still only one simulator. Is there a way to learn policies for all n robots simultaneously by using the same samples from the simulator? If yes, explain how. If no, explain why not. (4 points)

Yes. Keep track of n Q-value functions. Given a new sample update each value function independently and according to the above update equation where the ratio incorporates the corresponding transition model.

4 Probabilistic Inference (18 Points)

Consider the HMM graph structure shown below.



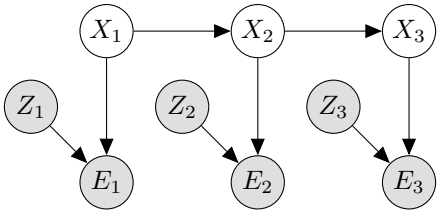
Recall the Forward algorithm is a two step iterative algorithm used to approximate the probability distribution $P(X_t|e_1, \ldots, e_t)$. The two steps of the algorithm are as follows:

Elapse Time $P(X_t|e_1, \ldots, e_{t-1}) = \sum_{x_{t-1}} P(X_t|x_{t-1})P(x_{t-1}|e_1, \ldots, e_{t-1})$

Observe $P(X_t|e_1, \ldots, e_t) = \frac{P(e_t|X_t)P(X_t|e_1,\ldots,e_{t-1})}{\sum_{x_t} P(e_t|x_t)P(x_t|e_1,\ldots,e_{t-1})}$

For this problem we will consider modifying the forward algorithm as the HMM graph structure changes. Our goal will continue to be to create an iterative algorithm which is able to compute the distribution of states, X_t, given all available evidence from time 0 to time t.

Consider the graph below where new observed variables, Z_i, are introduced and influence the evidence.



(a) State the modified Elapse Time update for $P(X_t|e_1, \ldots, e_{t-1}, z_1, \ldots, z_{t-1})$. (3 Points)

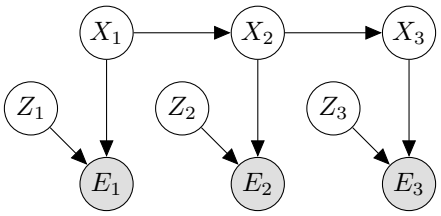
$P(X_t|e_1, \ldots, e_{t-1}, z_1, \ldots, z_{t-1}) = \sum_{x_{t-1}} P(X_t|x_{t-1})P(x_{t-1}|e_{1:t-1}, z_{1:t-1})$

(b) State the modified Observe update for $P(X_t|e_1, \ldots, e_t, z_1, \ldots, z_t)$. (3 Points)

$P(X_t|e_1, \ldots, e_t, z_1, \ldots, z_t) = \frac{P(X_t|e_{1:t-1},z_{1:t-1})P(z_t)P(e_t|z_t,X_t)}{\sum_{x_t} P(x_t|e_{1:t-1},z_{1:t-1})P(z_t)P(e_t|z_t,x_t)} = \frac{P(X_t|e_{1:t-1},z_{1:t-1})P(e_t|z_t,X_t)}{\sum_{x_t} P(x_t|e_{1:t-1},z_{1:t-1})P(e_t|z_t,x_t)}$

Here we need to incorporate the new Z_i variables. Since they are observed, we can assume that variable Z_t has value z_t.

Next, consider the graph below where the Z_i variables are unobserved.



(c) State the modified Elapse Time update for $P(X_t|e_1, \ldots, e_{t-1})$. (3 Points)

$P(X_t|e_1, \ldots, e_{t-1}) = \sum_{x_{t-1}} P(X_t|x_{t-1})P(x_{t-1}|e_{1:t-1})$

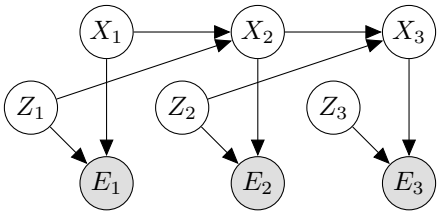
The Z_i's don't effect the time update.

(d) State the modified Observe update for $P(X_t|e_1, \ldots, e_t)$. (3 Points)

$P(X_t|e_1, \ldots, e_t) = \frac{P(X_t|e_{1:t-1}) \sum_{z_t} P(z_t)P(e_t|z_t, X_t)}{\sum_{x_t} P(x_t|e_{1:t-1}) \sum_{z_t} P(z_t)P(e_t|z_t, x_t)}$

Here we need to incorporate the new Z_i variables.

Finally, consider a graph where the newly introduced variables are unobserved and affect the transition model.



(e) State the modified Elapse Time update for $P(X_t, Z_t|e_1, \ldots, e_{t-1})$. (3 Points)

In the elapse time update, we want to get from $P(X_{t-1}, Z_{t-1}|e_{1:t-1})$ to $P(X_t, Z_t|e_{1:t-1})$.

$$\begin{aligned} P(X_t, Z_t|e_{1:t-1}) &= \sum_{x_{t-1}, z_{t-1}} P(X_t, Z_t, x_{t-1}, z_{t-1}|e_{1:t-1}) \\ &= \sum_{x_{t-1}, z_{t-1}} P(x_{t-1}, z_{t-1}|e_{1:t-1})P(X_t|x_{t-1}, z_{t-1}, e_{1:t-1})P(Z_t|X_t, x_{t-1}, z_{t-1}, e_{1:t-1}) \\ &= \sum_{x_{t-1}, z_{t-1}} P(x_{t-1}, z_{t-1}|e_{1:t-1})P(X_t|x_{t-1}, z_{t-1})P(Z_t) \end{aligned}$$

First line: marginalization, second line: chain rule, third line: conditional independence assumptions.

(f) State the modified Observe update for $P(X_t, Z_t|e_1, \ldots, e_t)$. (3 Points)

In the observation update, we want to get from $P(X_t, Z_t|e_{1:t-1})$ to $P(X_t, Z_t|e_{1:t})$.

$$\begin{aligned} P(X_t, Z_t|e_{1:t}) &\propto P(X_t, Z_t, e_t|e_{1:t-1}) \\ &\propto P(X_t, Z_t|e_{1:t-1})P(e_t|X_t, Z_t, e_{1:t-1}) \\ &\propto P(X_t, Z_t|e_{1:t-1})P(e_t|X_t, Z_t) \end{aligned}$$

First line: normalization, second line: chain rule, third line: conditional independence assumptions.

5 Self-Driving Cars and Backpropagation (10 Points)

You want to train a neural network to drive a car. Your training data consists of grayscale 64×64 pixel images. The training labels include the human drivers steering wheel angle in degrees and the human drivers speed in miles per hour. Your neural network consists of an input layer with $64 \times 64 = 4{,}096$ units, a hidden layer with 2,048 units, and an output layer with 2 units (one for steering angle, one for speed). You use the ReLU activation function for the hidden units and no activation function for the outputs (or inputs).

(a) Calculate the number of parameters (weights) in this network. You can leave your answer as an expression. (2 Points)

$4097 \times 2048 + 2049 \times 2$

(b) You train your network with the cost function $J = \frac{1}{2}|\mathbf{y} - \mathbf{z}|^2$. Use the following notation.

- $\mathbf{x}$ is a training image (input) vector with a 1 component appended to the end, $\mathbf{y}$ is a training label (input) vector, and $\mathbf{z}$ is the output vector. All vectors are column vectors.
- $r(\gamma) = \max\{0, \gamma\}$ is the ReLU activation function, $r'(\gamma)$ is its derivative (1 if $\gamma > 0$, 0 otherwise), and $r(\mathbf{v})$ is $r(\cdot)$ applied component-wise to a vector $\mathbf{v}$.
- $\mathbf{g}$ is the vector of hidden unit values before the ReLU activation functions are applied,
- $\mathbf{h} = r(\mathbf{g})$ is the vector of hidden unit values after the ReLU activation functions are applied (but we append a 1 component to the end of $\mathbf{h}$).
- V is the weight matrix mapping the input layer to the hidden layer, i.e., $\mathbf{g} = V\mathbf{x}$.
- W is the weight matrix mapping the hidden layer to the output layer, i.e., $\mathbf{z} = W\mathbf{h}$.

Derive $\partial J / \partial W_{ij}$. (3 Points)

$\partial J / \partial W_{ij} = (\mathbf{z} - \mathbf{y})^\top \partial \mathbf{z} / \partial W_{ij} = (\mathbf{z}_i - \mathbf{y}_i)\mathbf{h}_j$

(c) Write $\partial J / \partial W$ as an outer product of two vectors. $\partial J / \partial W$ is a matrix with the same dimensions as W; its just like a gradient, except that W and $\partial J / \partial W$ are matrices rather than vectors. (1 Points)

$\partial J / \partial W = (\mathbf{z} - \mathbf{y})\mathbf{h}^\top$

(d) Derive $\partial J / \partial V_{ij}$. (4 Points)

$\partial J / \partial V_{ij} = (\mathbf{z} - \mathbf{y})^\top \partial \mathbf{z} / \partial V_{ij} = (\mathbf{z} - \mathbf{y})^\top W \partial \mathbf{h} / \partial V_{ij} = ((\mathbf{z} - \mathbf{y})^\top W)_i r'(\mathbf{g}_i)\mathbf{x}_j$