Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

Examen
IIC2613 - Inteligencia Artificial
Primer Semestre, 2019

Tiempo: 2:30 hrs. Este examen es *sin* apuntes.

Pregunta 1

a) Un perceptrón multicapa debe tener al menos dos capas ocultas para poder resolver problemas no lineales. **(0.75 ptos.)**

Respuesta: Falso. Basta con una capa oculta para generar transformaciones Y combinaciones no lineales de las entradas

b) En un problema de clasificación multiclase, para que las salidas s_i de una CNN, tengan la forma de un histograma, o distribución de probabilidad, normalizada, basta con aplicar la siguiente función a cada una de ellas: **(0.75 ptos.)**

$$\sigma(s_i) = \frac{e^{s_i}}{\sum_j e^{s_j}}$$

Respuesta: Verdadero, la función σ es también conocida como *softmax*, y permite generar histogramas normalizados a partir de entradas reales.

c) Los *random forests* no puede resolver de manera perfecta problemas de clasificación no lineales. **(0.75 ptos.)**

Respuesta: Falso. Al combinar mediante votación las salidas de múltiples árboles, la superficie de decisión puede ser no lineal

d) La cantidad de vectores de soporte de un SVM lineal es siempre par. **(0.75 ptos.)**

Respuesta: Falso, es posible que sea par. En la formulación no existe ninguna restricción que fuerce esto. El caso canónico son tres vectores de soporte, donde el único vector de soporte de una de las dos clases se ubica sobre la recta que bisecta la recta que uno los vectores de soporte de la otra clase.

e) El programa dado por las reglas $\{p \leftarrow not\, q, q \leftarrow not\, p\}$ tiene un único modelo.

Respuesta: Falso. El programa tiene dos modelos: $\{p\}$ y $\{q\}$.

f) Si Π es un programa con n modelos que no menciona al predicado p, entonces el programa que resulta de agregar la regla $0\,\{p(a); p(b); p(c)\}\,2$ a Π tiene $7n$ modelos.

Respuesta: Verdadero. Debido a que Π no menciona a p, la nueva regla no puede hacer contradictorio a Π. La regla adicional en sí tiene 7 modelos pues corresponde a todas las formas que se pueden elegir 0 (1 modelo), 1 (3 modelos), o 2 (3 modelos) elementos de un conjunto de 3 elementos.

g) Suponga que tiene un problema de búsqueda en donde cada acción tiene costo igual a 1. Suponga además que la mejor heurística con la que cuenta es la heurística nula, es decir, una heurística que retorna 0 para todo estado. Entonces, para encontrar una solución óptima conviene ejecutar BFS (*breadth first search*) en vez de A*, pues BFS expande muchos menos estados.

Respuesta: Verdadero. Esto se debe a la condición de aceptación de BFS, que detecta un objetivo antes de que éste sea agregado a la frontera de búsqueda. A*, en cambio, solo lo detecta al objetivo cuando lo extrae de la frontera. Debido a esto A* puede demorar muchísimas más iteraciones que BFS porque en la práctica necesita expandir, adicionalmente, un nivel completo del árbol de búsqueda antes de encontrar el mismo objetivo que BFS retorna.

h) El número de estados en la lista *Closed* de DFS puede ser exponencial en el tamaño de la descripción del problema de búsqueda. (Para esta pregunta, suponga que los problemas de búsqueda se describen usando un estado final y acciones.)

Respuesta: Verdadero. Supongamos que la solución se encuentra sobre la última rama que DFS explora. En este caso, la lista de *Closed* constendrá, al momento de descubrir el objetivo, un número de estados similar a todo el espacio de búsqueda. A su vez, el espacio de búsqueda es típicamente exponencial en la descripción del problema.

Pregunta 2

Además de problemas de clasificación, el concepto de vectores de soporte también puede aplicarse a problemas de regresión. En base a esto, conteste las siguientes preguntas:

a) ¿Qué función podrían cumplir los vectores de soporte en una regresión de este tipo? **(2.0 ptos.)**

Respuesta: Los vectores de soporte indicarían el margen de tolerancia de error en la regresión. En otras palabras, cuán arriba o abajo de la recta pueden ubicarse los puntos.

b) Si se compara con una regresión lineal, ¿qué es lo que busca minimizar una regresión basada en vectores de soporte? (puede responder con palabras o con expresiones matemáticas) **(2.0 ptos.)**

Respuesta: Busca minimizar la suma de las distancias de los puntos que están fuera del margen, a la recta estimada.

c) ¿Para que servirían los *kernels* en este caso? **(2.0 ptos.)**

Respuesta: Tal como en un SVM, los kernels sirven para realizar transformaciones no lineales, lo que en este caso implica poder resolver problemas una regresión no lineal.

Pregunta 3

Se desea entrenar una CNN de 60 millones de parámetros, en un set de datos de clasificación de objetos (un objeto por imagen), que contiene 1000 objetos, pertenecientes a 100 categorías (10 imágenes por categoría). En base a esto, conteste las siguientes preguntas:

a) Indique y explique en detalle el problema principal al que se enfrentará el entrenamiento de esta red. **(2.0 ptos.)**

Respuesta: Dado el tamaño del set de datos, y la gran cantidad de parámetros de la red, el principal problema será el rápido sobreentrenamiento.

b) Considere ahora que tiene una CNN con la misma arquitectura, que ha sido preentrenada en un set de datos de clasificación de objetos (un objeto por imagen), que contiene un millón de objetos, pertenecientes a mil categorías (mil imágenes por categoría, ninguna repetida con respecto al set anterior). ¿Cómo podría utilizar esta red para entrenar una nueva red para el set de datos más pequeño? **(4.0 ptos.)**

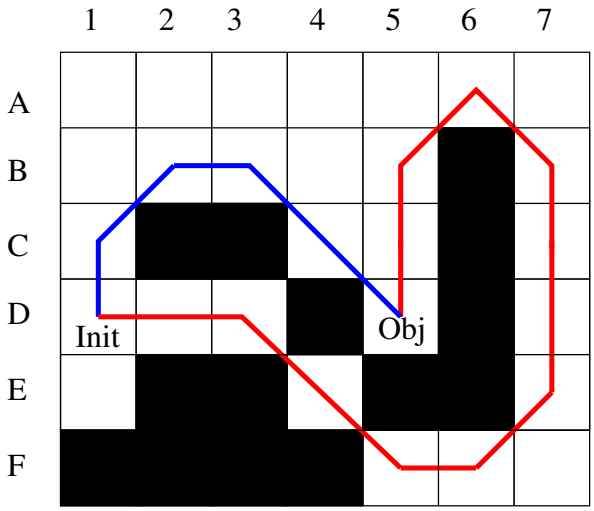
Respuesta: Acá se presentan dos posibles maneras de hacerlo. En la primera, se sustituye la capa de clasificación de la red preentrenada por una no entrenada con los tamaños adecuados para el nuevo set de datos. Luego esta nueva red se entrena manteniendo fijos los pesos de todas las capas, salvo los de la capa de clasificación. La segunda manera es muy similar, con la diferencia que en vez de mantener fijos los pesos de las capas que no hacen clasificación, se utiliza un *learning rate* bajo.

Pregunta 4

a) El contexto de esta pregunta es problemas de navegación de agentes en una grilla con obstáculos (tal como en la tarea 3 o ejemplos de clases). Supondremos que hay un solo agente que se puede mover en diagonal hacia una celda vecina vacía, a parte de horizontal y verticalmente. Supondremos que las cuatro movidas diagonales tienen costo $\sqrt{2}$ mientras que las horizontales y verticales tienen costo 1.

Invente un problema de navegación (con un solo agente) en el cual el algoritmo *greedy best-first search*, usado con la distancia euclidiana como heurística, retorna una solución cuyo costo es más del doble que el óptimo. **(3.0 ptos.)**

Respuesta: La grilla que muestra la figura muestra el camino óptimo en azul y el camino encontrado por el algoritmo greedy en rojo. El óptimo tiene costo $2 + 3\sqrt{2}$, mientras que el no óptimo tiene costo $8 + 5\sqrt{2}$, que claramente es más del doble del óptimo. Notemos que al iniciar la búsqueda tanto $C1$ como $E1$ quedan en la frontera. El nodo más cercano a G es $C1$ y su heurística es $\sqrt{17}$. Es fácil verificar que todas las celdas en el no camino óptimo tienen heurística menor a $\sqrt{17}$, y que por lo tanto son preferidas por el algoritmo greedy por sobre $C1$. De hecho, la celda en el camino rojo que está más lejos del objetivo, $A6$, tiene heurística $\sqrt{10}$.



b) El contexto de esta pregunta son los juegos de tablero con un adversario. Para limitar la profundidad de la búsqueda, Minimax utiliza una función de estimación que retorna una estimación del valor de un nodo . Suponga que cuenta con una función de estimación que al recibir un nodo n, retorna una tupla de dos valores (v, p), donde v es el valor estimado del nodo y p es un factor de incertidumbre que se mueve entre 0 y 1, donde 0 representa "ninguna incertidumbre" y 1 representa "máxima incertidumbre".

Escriba un pseudocódigo lo más detallado posible de una versión de Minimax que retorna la **mejor jugada** a ejecutar por el jugador *computador* y que, en vez de cortar la búsqueda usando un límite de profundidad, continúa haciendo búsqueda bajo un nodo si y solo si la incertidumbre es mayor que un valor T que se pasa como parámetro al algoritmo.

Suponga que puede usar el método `acciones_posibles(jugador)`, que se aplica sobre un nodo, y que retorna una lista de acciones posibles sobre ese nodo para un cierto `jugador`. Suponga que usa el valor 1 para identificar al jugador *computador* y -1 para identificar al jugador *adversario*. Suponga que el método `sucesor(accion)`, aplicado sobre un nodo, retorna el nodo que resulta de ejecutar la acción `accion`. Finalmente, suponga que el método `hoja()` retorna `True` si el nodo al cual se aplica es una hoja y `False` en caso contrario, y que el método `valor()` al ser aplicado sobre un nodo hoja, retorna el valor real (no una estimación) del nodo. **(4.0 ptos.)**

Respuesta:

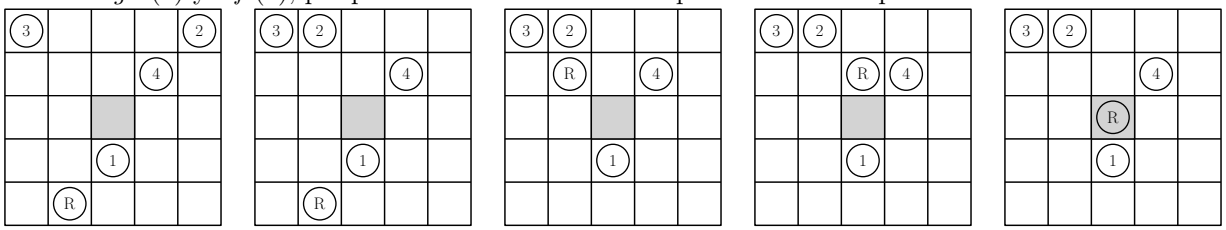
```
def Minimax(estado,t):
    acciones = estado.acciones_posibles(1)
    valores_hijos=[valor(estado.sucesor(a),t,-1) for a in acciones]
    return acciones[valores_hijos.index(min(valores_hijos))]
```

```
def valor(estado,t,jugador):
    if estado.hoja():
        return estado.valor()
    (val,p) = estado.evaluacion()
    if p < t:
        return val
    sucesores = [estado.sucesor(a) for a in estado.acciones_posibles()]
    valores = [valor(e,t,-jugador) for e in sucesores]
    if jugador==1:
        return min(valores)
    else:
        return max(valores)
```

Pregunta 5

Lunar Lockout es un puzle que se juega con 5 fichas ($\{R, 1, 2, 3, 4\}$), en un tablero de 5×5. Cada ficha se encuentra en una posición del tablero. Nunca hay fichas superpuestas. Para cada ficha f, las acciones disponibles son $up(f)$, $down(f)$, $left(f)$, $right(f)$ que desplazan a las fichas arriba, abajo, a la izquierda, y a la derecha, respectivamente. Cada acción aplicada sobre la ficha f tiene como efecto hacer avanzar a f todos los cuadros posibles en la dirección correspondiente, hasta que f "choca" con otra ficha, quedando f ubicada adyacente a la ficha con la cual chocó. Si una acción llevara a una ficha a salirse del tablero, entonces tal acción no es ejecutable. El objetivo es llevar a la ficha R hasta la celda central del tablero.

La figura de abajo muestra una solución a un problema. Note que en el estado inicial solo hay dos acciones ejecutables: $right(3)$ y $left(2)$, porque todas las demás harían que la ficha correspondiente se saliera del tablero.



estado inicial | después de $left(2)$ | después de $up(R)$ | después de $right(R)$ | después de $down(R)$

a) Dé una heurística admisible no trivial para este problema que solo pueda ser igual a 0 para el estado final. Suponga que cada acción tiene costo 1, independiente del número de cuadros que se desplaza la ficha. Describa su heurística en términos algorítmicos si le resulta más cómodo.

b) Demuestre que su heurística es admisible.

Respuesta: Hay muchas opciones. El máximo puntaje se otorga a una heurística al menos tan compleja como la que se describe abajo.

1. Inicializamos h en 0.
2. Primero, observamos que para llevar a R a la casilla central es necesario que haya una "ficha tope" adyacente a la casilla central. Si no la hay, incrementar h en 1. Esto mantiene admisibilidad porque al menos necesitamos una movida para esto.
3. Si R no se puede mover en la posición actual, entonces sumar 1 a h. La razón es que habrá que mover R de seguro en un estado no final. La admisibilidad está en que al menos una ficha deberá ser movida para que se genere el 'tope' que R necesita.
4. Si las dos condiciones anteriores se dieron, restar 1 a h. En efecto, con una sola movida se podrían haber logrado ambos objetivos (que R fuera movible y que hubiera una ficha ayacente al cuadrado central).
5. Si R está en línea horizontal o vertical con la celda central, sumar 1. En caso contrario, sumar dos. La admisibilidad se sigue porque al menos 1 acción es necesario ejecutar si estamos en línea y 2 acciones, si es que R no está en línea.

c) Diga cómo diseñaría un algoritmo de búsqueda heurística eficiente para este problema con la siguientes 4 características. (1) El algoritmo no retorna una solución, sino que una secuencia de soluciones $s_1, s_2, \ldots, s_n$. (2) El costo de la secuencia es decreciente (es decir, si $i < j$, entonces $costo(s_i) > costo(s_j)$. (3) El costo de la primera solución no excede en w veces el óptimo, donde w es un parámetro del algoritmo. (4) s_n es una solución óptima.

Respuesta: Podríamos llamar a Weighted A* repetidamente, sobre el mismo problema, con una secuencia de pesos que comience en w y termine en 1. (Por ejemplo, si $w = 1+\epsilon$ podríamos usar $\{1+\epsilon, 1+\frac{3}{4}\epsilon, 1+\frac{1}{2}\epsilon, 1+\frac{1}{4}\epsilon, 1\}$. Para garantizar que el costo es decreciente, al sacar un nodo n de la lista *Open* lo descartamos si es que $g(n)+h(n)$ es mayor que el costo de la solución.