Laboratorio 4: Aprendizaje por Refuerzo Profundo

Curso: Inteligencia Artificial

---

1 Anotaciones

En este laboratorio, vamos a construir un automóvil autónomo desde cero, en un mapa 2D, utilizando el modelo de DQN y el framework Kivy.

1.1 Entorno

Aquí, tenemos mucho más que definir que solo estados, acciones y recompensas. Construir un automóvil autónomo es un problema seriamente complejo y se realizará en un mapa 2D dentro de una aplicación web Kivy.

Kivy es un framework de Python gratuito y de código abierto, que se utiliza para el desarrollo de aplicaciones como juegos o en realidad cualquier tipo de aplicación móvil. Todo el entorno de este proyecto está construido con Kivy de principio a fin.

1.2 Definiendo el objetivo

Nuestro objetivo es construir un automóvil autónomo.Pero, ¿cómo vamos a formalizar ese objetivo en términos de IA y aprendizaje por refuerzo?. En este laboratorio, vamos a formalizar un objetivo: entrenar al automóvil autónomo para realizar viajes de ida y vuelta entre un aeropuerto y el centro de la ciudad. Tan pronto como llegue al aeropuerto, deberá dirigirse al centro y tan pronto como llegue al centro, deberá dirigirse al aeropuerto. Más que eso, debería poder hacer estos viajes de ida y vuelta a lo largo de cualquier camino que conecte estos dos lugares. También debería poder hacer frente a cualquier obstáculo a lo largo de ese camino que deba evitar

1.3 Parámetros

Antes de definir los estados de entrada, las acciones de salida y las recompensas, debes configurar todos los parámetros del mapa y el automóvil que formará parte de su entorno. Las entradas, salidas y recompensas son todas funciones de estos parámetros. Vamos a enumerarlos todos, usando los mismos nombres que en el código, para que se pueda entender fácilmente el archivo `map.py`:

1. $angle$: el ángulo entre el eje x del mapa y el eje del automóvil
2. $Rotation$: última rotación que realiza el coche
3. $pos = (self.car.x, self.car.y)$: La posición del automóvil ($self.car.x$ es la coordenada x del automóvil, $self.car.y$ es la coordenada y del coche)
4. $velocity = (velocity\_x, velocity\_y)$: el vector de velocidad del automóvil
5. $sensor1 = (sensor1\_x, sensor1\_y)$: la posición del primer sensor
6. $sensor2 = (sensor2\_x, sensor2\_y)$: la posición del segundo sensor

7. $sensor3 = (sensor3\_x, sensor3\_y)$: la posición del tercer sensor
8. $signal1$: la señal recibida por el sensor 1
9. $signal2$: la señal recibida por el sensor 2
10. $signal3$: la señal recibida por el sensor 3.

Los últimos parámetros son importantes porque son las últimas piezas que necesitamos para revelar el vector de estado de entrada final:

1. $goal\_x$: la coordenada x del objetivo (que puede ser el aeropuerto o el centro)
2. $goal\_y$: la coordenada y del objetivo (que puede ser el aeropuerto o el centro)
3. $xx = (goal\_x - self.car.x)$: La diferencia de coordenadas x entre el objetivo y el auto
4. $yy = (goal\_y - self.car.y)$: La diferencia de coordenadas y entre el objetivo y el auto
5. $Orientation$: el ángulo que mide la dirección del coche con respecto al objetivo.

1.4 Los estados de entrada

La orientación es una entrada única que nos dice directamente si estamos apuntando en la dirección correcta, hacia el objetivo. Si tenemos esa orientación, no necesitamos las coordenadas de posición del coche para navegar hacia el objetivo. Tenemos el primer elemento de nuestro estado de entrada: la orientación.

Pero eso no es suficiente. Recuerda que también tenemos otro objetivo, o, debería decir limitación. Nuestro automóvil debe permanecer en la carretera y evitar cualquier obstáculo a lo largo de esa carretera. En el estado de entrada, necesitamos información que le diga a la IA si está a punto de salirse de la carretera o chocar contra un obstáculo. La solución son los sensores.

Las señales de estos sensores ya están codificadas en tres variables: $signal1$, $signal2$ y $signal3$. Estas señales le dirán a la IA si está a punto de chocar con algún obstáculo o de salirse de la carretera, ya que la carretera está delimitada por arena. Ese es el resto de la información que necesita para tu estado de entrada. Con estos cuatro elementos, $signal1$, $signal2$, $signall3$ y $orientatin$, tienes todo lo que necesitas para poder conducir de un lugar a otro, permaneciendo en la carretera y sin chocar con ningún obstáculo. En conclusión, este será el estado de entrada en cada momento:

```
Estado de entrada  = (orientation, signal1, signal2, signal3)
```

1.5 Acciones de salidas

Dado que está construyendo un automóvil autónomo, las acciones de salidas pueden ser: avanzar, girar a la izquierda o girar a la derecha y esta afirmación no solo es intuitiva, sino que se alinea extremadamente bien con nuestra elección de estados de entrada. Contienen la variable *orientation* que nos dice si estamos apuntando en la dirección correcta hacia el objetivo. Al mismo tiempo, si alguna de las señales detecta arena alrededor del automóvil, el automóvil girará hacia la izquierda o hacia la derecha para evitarlo. Las tres acciones posibles de avanzar, girar a la izquierda y girar a la derecha tienen sentido lógico con los estados de entrada, restricción y objetivo que tenemos.

1.6 Recompensa

Para definir el sistema de recompensas, tenemos que responder a las siguientes preguntas:

- ¿En qué casos le damos a la IA una buena recompensa?
- ¿Qué tan bueno para cada caso?
- ¿En qué casos le damos a la IA una mala recompensa?
- ¿Qué tan malo para cada caso?

Para responder a estas preguntas, simplemente debemos recordar cuál es el objetivo y las limitaciones: El objetivo es realizar viajes de ida y vuelta entre el aeropuerto y el centro de la ciudad.Las limitaciones son permanecer en la carretera y evitar obstáculos si los hay. En otras palabras, la restricción es mantenerse alejado de la arena.

Por lo tanto, en función de este objetivo y las limitaciones, las respuestas a nuestras preguntas anteriores son:

1. Le damos a la IA una buena recompensa cuando se acerca al destino.
2. Le damos a la IA una mala recompensa cuando se aleja más del destino.
3. Le damos a la IA una mala recompensa si está a punto de pisar arena.

Para responder a la segunda parte de cada pregunta, qué tan buena y qué tan mala debe ser la recompensa para cada caso, podemos asumir que podemos castigar más al coche cuando comete errores que en recompensarlo cuando lo hace bien. Esto funciona bien en el aprendizaje por refuerzo. En ese sentido, aquí están las recompensas que daremos en cada caso:

1. La IA obtiene una mala recompensa de -1 si conduce sobre arena.
2. La IA obtiene una mala recompensa de -0.2 si se aleja del destino.
3. La IA obtiene una buena recompensa de 0.1 si se acerca al destino.

2 Solución de Inteligencia Artificial

Indiquemos los pasos del proceso profundo de Q-learning, que adaptamos a nuestra aplicación para vehículos autónomos.

Inicialización:

1. La memoria de la repetición de la experiencia se inicializa en una lista vacía, denominada memoria en el código.
2. Se establece el tamaño máximo de la memoria, llamado capacidad en el código.

En cada tiempo t, la IA repite el siguiente proceso, hasta el final de la época:

1. La IA predice los Q valores del estado actual s_t. Por lo tanto, si se pueden jugar tres acciones se obtienen tres Q valoresprevistos.
2. La IA realiza una acción seleccionada por el método softmax: $a_t = \text{softmax}_a Q(s_t, a)$.
3. La IA recibe una recompensa $R(s_t, a_t)$ que es una de $-1, -0,2$ o $+0,1$.
4. La IA alcanza el siguiente estado s_{t+1}, que se compone de las siguientes tres señales de los tres sensores, más la orientación del automóvil.
5. La IA agrega la transición (s_t, a_t, r_t, s_{t+1}) a la memoria.
6. La IA toma un lote aleatorio $B \subset M$ de transiciones. Para todas las transiciones $(s_{t_B}, a_{t_B}, r_{t_B}, s_{t_B+1})$ del lote aleatorio B:
   - La IA obtiene las predicciones: $Q(s_{t_B}, a_{t_B})$
   - La IA obtiene los objetivos: $R(s_{t_B}, a_{t_B}) + \gamma \max_a (Q(s_{t_B}, a_{t_B}))$
   - La IA calcula la pérdida entre las predicciones y los objetivos en todo el lote B

$$\text{Pérdida} = \frac{1}{2} \sum_B \left(R(s_{t_B}, a_{t_B}) + \gamma \max_a \left(Q(s_{t_B+1}, a) \right) - Q(s_{t_B}, a_{t_B}) \right)^2$$

   - Finalmente, la IA retropropaga este error de pérdida en la red neuronal y a través del descenso del gradiente estocástico, actualiza los pesos de acuerdo con cuánto contribuyeron al error de pérdida.

3 Implementación

La implementación de vehículos autónomos se compone de tres archivos Python:

1. *car.kv* que contiene los objetos Kivy (forma rectangular del automóvil y los tres sensores)
2. *map.py*, que construye el entorno (mapa, automóvil, estados de entrada, acciones de salida, recompensas)
3. *deep_q_learning.py*, que crea y entrena la IA a través de Q-aprendizaje.

4 Utilización

1. Dentro de la terminal, ingresamos el siguiente comando: `conda create -n [nombre-entorno] python=3.6`. Este comando crea un entorno virtual con Python 3.6 y otros paquetes instalados.
2. Luego, vamos a activar el entorno virtual, lo que significa que vamos a entrar en él para instalar PyTorch y Kivy. Para activar el entorno, ingresaremos el siguiente comando: `conda activate [nombre-entorno]`.
3. Instalamos pytorch y kivy con conda. Por ejemplo para instalar kivy realizamos lo siguiente: `conda install -c conda-forge/label/cf201901 kivy`.
4. Luego ingresemos el comando final, `python map.py`. Durante el primer minuto más o menos, tu automóvil autónomo explorará sus acciones realizando movimientos sin sentido, es posible que lo vea girando. Después de cada 100 movimientos, los pesos dentro de la red neuronal de la IA se actualizan y el automóvil mejora sus acciones para obtener mayores recompensas.

5 Ejercicios

1. Explica estas líneas de código:

```
def learn(self, batch_states, batch_actions, batch_rewards,batch_next_states):
  batch_outputs = self.model(batch_states).gather(1, batch_actions.unsqueeze(1)).squeeze(1)
  batch_next_outputs = self.model(batch_next_states).detach().max(1)[0]
  batch_targets = batch_rewards + self.gamma * batch_next_outputs
  td_loss = F.smooth_l1_loss(batch_outputs, batch_targets)
  self.optimizer.zero_grad()
  td_loss.backward()
  self.optimizer.step()
```

2. Dibuja algunos obstáculos en el mapa para ver si el coche los esquiva. Después de unos minutos más de entrenamiento se puedo ver claramente el coche evitando los obstáculos. Modifica algunos parámetros si fuese necesario.