Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

REDES NEURONALES ARTIFICIALES
AGO-DIC 21

Redes neuronales - Producto integrador

Nombre:	**Matricula:**	**Carrera:**
Luis Daniel García Leal	1857391	ITS

Nombre del profesor:

JOSE ARTURO BERRONES SANTOS

Semestre agosto – diciembre 2021

Días de la clase y Hora: Jueves (V4-V6)

San Nicolás de los Garza, N.L. Fecha: 25/11/2021

Resumen

El problema del CartPole es uno de los entornos de aprendizaje por refuerzo más simples, descrito por primera vez por Barto, Sutton y Anderson en 1983. En este entorno, un poste está unido a un carro que se puede mover a lo largo de una pista sin fricción. La elección de este como mi proyecto de esta materia fue por su simplicidad de realizar y porque este aplica técnicas vistas en esta clase, creo que es un ejemplo bastante practico. Utilizaremos las herramientas de OpenAI para implementar este problema.

Características del problema a estudiar y los datos

Las principales características del entorno se resumen a continuación:

Descripción: Un poste está unido por una junta no accionada a un carro, que se mueve a lo largo de una pista sin fricción. El sistema se controla aplicando una fuerza de +1 o -1 al carro. El péndulo comienza erguido, y el objetivo es evitar que se caiga. Se proporciona una recompensa de +1 por cada paso de tiempo que el poste permanece erguido. El episodio termina cuando el poste está a más de 15 grados de la vertical, o el carro se mueve a más de 2,4 unidades del centro.

1. Objetivo: Mantenga el poste en posición vertical durante n cantidad de pasos de tiempo.

2. Estado: Una matriz de longitud a definir que representa: [posición del carro, velocidad del carro, ángulo del polo, velocidad angular del polo]. Por ejemplo, [−0.034, 0.032, −0.031, 0.036].

3. Acción: Un entero, ya sea 0 para mover el carro una distancia fija a la izquierda, o 1 para mover el carro una distancia fija a la derecha.

4. Recompensa: +1 por cada paso de vez que el poste permanece erguido.

5. Terminación: Cuando el poste se cae (más de 12 grados desde la vertical), o cuando el carro se mueve fuera de la pantalla, o cuando se alcanza el paso de tiempo máximo de 200

Aprendizaje:

Arquitectura de red: Perceptrón multicapa con una capa oculta de unidades y funciones de activación.

Optimizador: El optimizador es Adam

Objetivos

Instrucciones

El alumno deberá proponer un problema de análisis de datos y resolverlo usando cualquiera de las técnicas computacionales estudiadas en el curso. Deberá entregarse PDF con las siguientes secciones: Resumen, Características del problema a estudiar y los datos, Objetivos, Metodología (con explicación de la técnica o técnicas computacionales empleadas) y Resultados.

Contenido

Conocer a un nivel cualitativo el estado del arte en cuanto al poder transformador de los grandes datos y la inteligencia computacional en la industria con aplicaciones como manufactura, planeación y pronóstico de producción/demanda y analítica de negocios.

Criterio de Desempeño

Mostrar entendimiento claro de los conceptos de inteligencia artificial y sus aplicaciones contemporáneas.

Actividad de Aprendizaje

Lectura y generación de resumen crítico (con interpretación del alumno) de artículos selectos de divulgación de los sistemas inteligentes aplicados en la industria.

Metodología (con explicación de la técnica o técnicas computacionales empleadas)

Herramientas de desarrollo

Python: Es un lenguaje de programación interpretado, orientado a objetos de alto nivel y con semántica dinámica. Su sintaxis hace énfasis en la legibilidad del código, lo que facilita su depuración y, por tanto, favorece la productividad. Ofrece la potencia y la flexibilidad de los lenguajes compilados con una curva de aprendizaje suave. Esta es la razón por la que se incluye como lenguaje de programación básico en un Máster en Data Science.

Numpy: NumPy es una librería de Python que da soporte al usuario y le permite crear vectores y matrices grandes multidimensionales, junto con una gran colección de funciones matemáticas de alto nivel para operar con ellas.

Tensorflow: TensorFlow es una biblioteca de código abierto para aprendizaje automático a través de un rango de tareas, y desarrollado por Google para satisfacer sus necesidades de sistemas capaces de construir y entrenar redes neuronales para detectar y descifrar patrones y correlaciones, análogos al aprendizaje y razonamiento usados por los humanos.

Gym: Gym es una biblioteca Python de código abierto para desarrollar y comparar algoritmos de aprendizaje por refuerzo al proporcionar una API estándar para comunicarse entre algoritmos de aprendizaje y entornos, así como un conjunto estándar de entornos compatibles con esa API. Desde su lanzamiento, la API de Gym se ha convertido en el estándar de campo para hacer esto.

Matplotlib: Es una librería enfocada a la generación de gráficos a partir de datos contenidos en listas o arrays en el lenguaje de programación Python y su extensión matemática NumPy.

Spyder: Spyder es un entorno de desarrollo integrado multiplataforma (IDE) de código abierto para la programación científica en el lenguaje Python. Spyder se integra con una serie de paquetes prominentes en la pila científica de Python, incluyendo NumPy, SciPy, Matplotlib, pandas, IPython, SymPy y Cython, así como otro software de código abierto. Se libera bajo la licencia del MIT.

Para comenzar con este código vamos a instalar las bibliotecas necesarias para que nuestro entorno de programación funcione.

```
"""
!pip install tensorflow == 2.3.0
!pip install gym
!pip install keras
!pip install keras-rl2
!pip install gym[all]
```

Después de asegurarnos que de que contamos con los recursos necesarios importaremos las librerías gym y random para instanciar nuestro algoritmo de cart-pole, en este caso utilizamos la librería random para generar un numero aleatorio que funciona básicamente como la variable de entrada y que determinara la dirección de nuestro sujeto. En esta parte también instanciamos los estados del entorno y extraemos las cantidades de acciones siendo estas 2. (Lineas de código 16 a 18)

```
import gym
import random

env = gym.make('CartPole-v0')
states = env.observation_space.shape[0]
actions = env.action_space.n

print(actions)
```

Primero lo que vamos a hacer es renderizar el entorno mediante una pestaña y se podrá ver como el carrito interactúa con lo que seria la vara durante la cantidad de episodios que queramos que se muestre, en este caso son 10. Random.choice representa si el carro se moverá a la izquierda o a la derecha.

Al renderizarlo, se podrá observar el agente aún no está entrenado, por lo que no puede dar más de un par de pasos. Pronto exploraremos algunas de las estrategias que mejorarán drásticamente el rendimiento. Después de dicha acción se obtendrá una recompensa dependiendo si se tomó una decisión correcta, esta variable es la que le permitirá nuestro algoritmo obtener información sobre lo que debería hacer está siendo la variable score. (Líneas de Código 22 a 33)

```
episodes = 10
for episode in range(1, episodes+1):
    state = env.reset()
    done = False
    score = 0

    while not done:
        env.render()
        action = random.choice([0,1])
        n_state, reward, done, info = env.step(action)
        score += reward
    print('Episode:{} Score{}'.format(episode, score))
```

Aquí es donde comenzamos a implementar un modelo de aprendizaje profundo, pero para ello será necesario importaremos Numpy que nos permitirá utilizar Arrays, después importaremos un modelo de Keras secuencial API, importaremos las capas de nodos densas y simples y finalmente importaremos nuestro optimizador Adam. Los optimizadores son algoritmos o métodos utilizados para cambiar los atributos de su red neuronal, como los pesos y la tasa de aprendizaje, con el fin de reducir las pérdidas. La

forma en que debe cambiar sus pesos o tasas de aprendizaje de su red neuronal para reducir las pérdidas está definida por los optimizadores que utiliza. (Líneas de Código 35 a 38).

Creamos una función llamada build model, en esta función extraeremos los estados y nuestras acciones “def build_model(states, actions): ”, primero instanciamos nuestro modelo que en este caso es secuencial (Un módulo secuencial es una clase contenedora o contenedora que extiende el nn. Clase base de módulos y nos permite componer módulos juntos. Podemos componer cualquier nn. Módulo con en cualquier otro nn. Módulo.). (Líneas de Código 41).

Después introducimos los nodos simples, el cual contiene como dato de entrada los estados anteriormente mencionados, añadimos dos nodos densos/profundos para que sea posible implementar nuestro modelo de Deep learning que contiene una activación relu y nuestro ultimo nodo pasamos lo que serian las acciones de la inteligencia artificial. (Líneas de Código 40 a 45).

Después de declarar nuestra función recibiremos los datos y comenzaremos a construir nuestro modelo. (Líneas de Código 47 a 48).

```
35  import numpy as np
36  from tensorflow.keras.models import Sequential
37  from tensorflow.keras.layers import Dense, Flatten
38  from tensorflow.keras.optimizers import Adam
39
40  def build_model(states, actions):
41      model = Sequential()
42      model.add(Flatten(input_shape=(1,states)))
43      model.add(Dense(24,activation='relu'))
44      model.add(Dense(actions, activation='linear'))
45      return model
46
47  model=build_model(states, actions)
48  model.summary()
```

Importaremos nuestras dependencias Keras las cuales son DQNAgent, una política de BoltzmannQPolicy y una memoria secuencial. Lo que le permitirá a nuestro agente almacenar datos.

El agente DQN se puede utilizar en cualquier entorno que tenga un espacio de acción discreto. En el corazón de un agente DQN se encuentra un QNetwork, un modelo de red neuronal que puede aprender a predecir QValues (rendimientos esperados) para todas las acciones, dada una observación del entorno. Utilizaremos tf_agents.networks. para crear un QNetwork. (Líneas de Código 50 a 52).

Después construiremos una función que se encargara de desarrollar nuestro agente, instanciamos nuestra política y memoria, establecemos parámetros (Líneas de Código 55 a 60). Utilizamos el agente para entrenar a nuestro modelo y finalmente lo alimentamos. (Líneas de Código 62).

Instanciamos nuestras funciones, lo compilamos con nuestro optimizador con sus respectivos parámetros como el modelo, la memoria y la política. Usamos la función fit para definir los números de pasos, visualizar entre otras cosas y finalmente guardamos nuestro nuevo modelo mediante dqn.save_weights. (Líneas de Código 63 a 69).

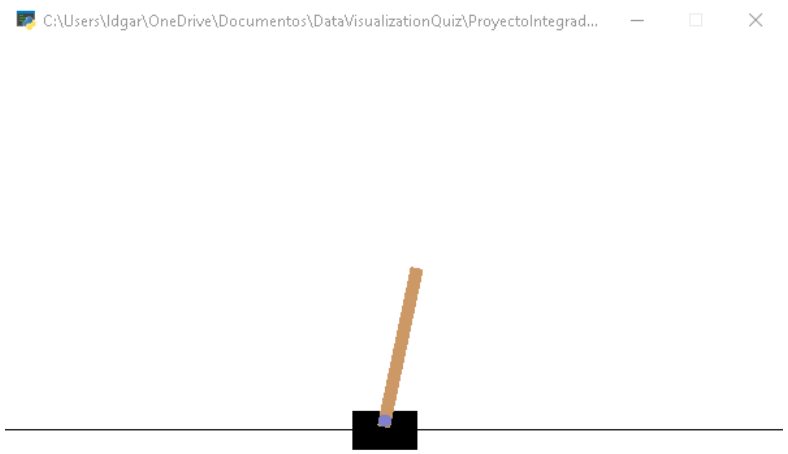
```
50  from rl.agents import DQNAgent
51  from rl.policy import BoltzmannQPolicy
52  from rl.memory import SequentialMemory
53
54
55  def build_agent(model, actions):
56      policy = BoltzmannQPolicy()
57      memory = SequentialMemory(limit=50000,window_length=1)
58      dqn = DQNAgent(model=model,memory=memory,policy=policy,nb_actions=actions,
59                     nb_steps_warmup=10, target_model_update=1e-2)
60      return dqn
61
62  dqn = build_agent(model, actions)
63  dqn.compile(Adam(lr=1e-3), metrics=['mae'])
64  dqn.fit(env, nb_steps= 50000, visualize=False, verbose=1)
65
66  scores = dqn.test(env, nb_episodes=100, visualize = False)
67  print(np.mean(scores.history['episode_reward']))
68
69  dqn.save_weights('dqn_weights.h5f', overwrite=True)
```

Ahora con nuestro modelo entrenado podemos utilizarlo y mostrarlo para propósitos de comparación, la metodología para hacer eso se encuentra en las siguientes líneas de Código, hacer esto nos servirá para comparar que tan efectivo fue el entrenamiento implementado. (Líneas de Código 72 a 84).

```
72  del model
73  del dqn
74  del env
75
76  env= gym.make('CartPole-v0')
77  actions= env.action_space.n
78  states= env.observation_space.shape[0]
79  model = build_model(states, actions)
80  dqn = build_agent(model, actions)
81  dqn.compile(Adam(lr=1e-3), metrics=['mae'])
82
83  dqn.load_weights('dqn_weights.h5f')
84  _ = dqn.test(env, nb_episodes=5, visualize = True)
```

Resultados.

Modelo: Como se puede observar en las siguientes imágenes la inteligencia artificial suele tener un comportamiento más erróneo y espontaneo, también puede observarse que la recompensa que obtiene por cada juego no es nula, pero tampoco es demasiada ni es consistente.




```
Episode:1 Score15.0
Episode:2 Score26.0
Episode:3 Score20.0
Episode:4 Score11.0
Episode:5 Score27.0
Episode:6 Score10.0
Episode:7 Score9.0
Episode:8 Score36.0
Episode:9 Score26.0
Episode:10 Score16.0
Model: "sequential_1"
```

Modelo entrenado con técnicas de entrenamiento Deep learning: En comparación al modelo anterior como se puede apreciar en las siguientes imágenes el sistema tiene una mayor estabilidad con lo que seria la vara, los movimientos que este realiza son mas precisos y calculados, todo esto fue posible gracias al modelo explicado anteriormente, a continuación puede verse un output de nuestro modelo y también la cantidad de episodios en la cual se puede observar que en la mayoría de los casos este obtiene una mayor recompensa, esto es gracias a la cantidad de capas que tiene nuestro modelo y a la capacidad del modelo de aprender de sus errores.

Output de modelo:

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_1 (Flatten)          (None, 4)                 0
_________________________________________________________________
dense_3 (Dense)              (None, 24)                120
_________________________________________________________________
dense_4 (Dense)              (None, 24)                600
_________________________________________________________________
dense_5 (Dense)              (None, 2)                 50
=================================================================
Total params: 770
Trainable params: 770
```

```
Episode 97: reward: 200.000, steps: 200
Episode 98: reward: 200.000, steps: 200
Episode 99: reward: 200.000, steps: 200
Episode 100: reward: 200.000, steps: 200
200.0
Testing for 5 episodes ...
Episode 1: reward: 200.000, steps: 200
Episode 2: reward: 200.000, steps: 200
Episode 3: reward: 200.000, steps: 200
Episode 4: reward: 200.000, steps: 200
Episode 5: reward: 200.000, steps: 200
```