PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA
IIC2613 - INTELIGENCIA ARTIFICIAL

TAREA 4

Fecha Máxima de Entrega: 19/07

Objetivo y Regla importante

En esta tarea veremos el uso práctico de redes neuronales artificiales o ANN, y resolveremos un problema de aprendizaje reforzado. Al igual que en la tarea anterior se requerirá de múltiples librerías especializadas para las que les entrengamos un resumen para su uso en esta tarea.

Antes de seguir avanzando, y como ha sido costumbre en el curso vamos a repasar nuestra regla de integridad, con una actualización para esta tarea. Estamos conscientes que algunas respuestas a las preguntas de esta tarea se pueden encontrar en Internet, en libros, o en conversaciones con tus compañeros, de hecho esta tarea tiene un ítem de investigación. Como esta tarea tiene un fin pedagógico, la regla de integridad académica que pedimos que cumplas es la siguiente:

Toda respuesta a una pregunta teórica debe ser escrita por ti, individualmente. Esto significa que al momento de editar la respuesta, no debes copiar el material hecho por otra persona. ***Queremos tu comprensión y entendimiento de las cosas que estudies y aprendas****. Por lo tanto, las definiciones y análisis que presentes deben venir de tu propio razonamiento.*
También es importante dejar claro que sabemos que estás utilizando material hecho por otras personas, y debes reconocerlo. ***Es por eso que te pediremos que cites tus fuentes en esta oportunidad****. Tienes dos formas de citar. La primera es bastante indirecta, y solamente exige que menciones a tu fuente en tu bibliografía. Harás esto cuando hayas aprendido conceptos, los hayas interiorizado, y estés dando tu interpretación. La segunda es más directa, utilizarás las comillas "" para parafrasear algo que se le ocurrió a otra persona, seguido de un paréntesis con la referencia de dónde se encuentra dicho texto. Solamente debes parafrasear cuando quieres darle fuerza a un argumento, jamás para contestar algo que se te pregunte a ti, pues tal respuesta, si bien te evita el plagio, tampoco te da puntaje. Las citas de publicaciones científicas las puedes poner en cualquier formato de amplia aceptación como APA, ICONTEC e IEEE, por nombrar algunos. Las citas a sitios web las debes hacer referenciando el link del sitio, y la fecha de la que has tomado la referencia. Por favor, no cites a tus compañeros, aprende de ellos, sé crítico con sus opiniones, y entrega tu propio trabajo original.*
Esta misma regla también se cumple respecto del código que entregues.

Tu entrega deberá incluir un archivo llamado **INTEGRIDAD.txt**, en donde afirmas que has leído, entendido y respetado la regla de arriba. De no cumplirse esto, **tu tarea no será corregida, y en consecuencia obtendrás la nota mínima, tal como si no hubieses entregado**. Recuerda subir todo el material relacionado a tu tarea en tu repositorio personal de GitHub a más tardar la fecha de entrega a las 23:59, para esto deben crear el directorio `T4/` que contenga todos los archivos y subdirectorios especificados.

1 Librerías

La actividad la realizaremos utilizando códigos en lenguaje de programación Python. Utilizaremos diversas librerías de aprendizaje automático. Se les entregará un archivo template en el que podrán encontrar algunos scripts, en especial para uso del ambiente de simulación. Por lo demás aquí veremos un resumen de algunas

clases, métodos y funciones de interés.

Es importante considerar con cuidado el aprendizaje de estas librerías, pues en algunos casos su utilización toma tiempo. Se presentan dos librerías, Keras y Pytorch, para el trabajo de redes neuronales. Sea cuidadoso al elegir cuál utilizará, ya que una es más fácil de utilizar y aprender a usar que la otra, pero la segunda le puede servir más en el futuro si desea seguir con esta línea de aprendizaje.

1.1 Pandas

A nuestro repertorio de métodos conocidos agregaremos dos que nos facilitarán la identificación de elementos únicos.

El método `nunique` nos dirá en cada serie el número de elementos básicos que se pueden encontrar. Algunos valores etiquetados en categorías no numéricas pueden ser trabajados si las cantidades de dichas categorías son pequeñas. El siguiente es un ejemplo que nos mostraría cuántos elementos básicos hay en cada columna.

```
for col in df.columns:
    print(col, df[col].nunique())
```

El método `unique` nos desplegará en cada serie los elementos básicos que componen la serie, o elementos que se repiten. El siguiente es un ejemplo que nos mostraría estos bloques generadores de la serie en cada columna.

```
for col in df.columns:
    print(col, df[col].unique())
```

1.2 Keras

Keras es una librería de redes neuronales, desarrollada por François Chollet, y posteriormente agregada sobre Tensorflow (una librería más potente con el mismo fin). Tiene la particularidad de ser fácil de utilizar, con una rápida curva de aprendizaje, y relativamente rápida redacción de código. Sacrifica un poco de control y acceso a variables internas por este empaquetamiento extra. La gran mayoría de entradas y salidas se trabajan como un arreglo de NumPy.

Si bien en esta oportunidad no es necesario el uso de GPU, keras puede detectar automáticamente si se encuentra disponible este hardware. De todas formas puede utilizar el siguiente código para verificar si Keras detecta o no la GPU.

```
from keras import backend as K
K.tensorflow_backend._get_available_gpus()
```

Algunas de las clases y funciones relevantes para esta tarea se importarán de la siguiente manera:

```
import keras
from keras.layers import Dense
from keras.models import Sequential
from keras.utils.np_utils import to_categorical
```

A continuación revisaremos algunas particularidades de las clases, métodos y funciones disponibles:

- `model = Sequential()`: Se inicializa una instancia de la clase `Sequential`. Esta clase representará una red neuronal, con sus capas ordenadas en una secuencia.
- `model.add(layer)`: Método para agregar una nueva capa a la red. El orden importa, pues es el que seguirá el flujo de datos, o mapa de activación. La primera capa corresponde a la entrada, mientras que la última corresponde a la salida.

- `layer = Dense(n, input_shape=shape, activation=activation_func)`:
  La clase `Dense` representa una capa densa o lineal, la única necesaria en esta tarea. El parámetro `n` hace referencia a la cantidad de neuronas presentes en la capa. El parámetro `input_shape` especifica la forma del tensor de entrada, con `shape` normalmente una tupla que indica la forma de dicho tensor. Este parámetro solo se tiene que especificar en la capa de entrada, con la forma de los datos de entrada (normalmente no es necesario indicar la cantidad de filas en una matriz X de $m \times n$, por lo que la tupla de entrada sería $(m,)$), ya que la clase `Sequential` utiliza la información de la capa anterior para conectar la entrada. El parámetro `activation` especificará qué función de activación aplicará la capa antes de pasar los datos a la siguiente capa. Si no se especifica no se le aplica una función de activación a los datos. Algunos valores que puede tomar `activation_func` son [`'elu'`, `'relu'`, `'tanh'`, `'sigmoid'`, `'softmax'`].

- `model.Summary()`: Método útil para ver un resumen de la red y de los parámetros que tiene cada capa.

- `model.compile(loss=None, optimizer='rmsprop', metrics=None)`: Método que generaá el grafo de cómputo para el entrenamiento. La función de pérdida `loss` se puede especificar como un string (`'categorical_crossentropy'` o `'mse'`, por ejemplo), o se puede definir una clase de `keras.losses`. El `optimizer` verificará la forma en que se actualizan los pesos con el gradiente. Al igual que `loss` se puede espeficicar con un string (`'adam'` o `'sgd'`, por ejemplo) o especificar una clase con `keras.optimizers`. Finalmente, en `metrics` se puede especificar la o las métricas que serán utilizadas de la misma forma, bien pudiendo ser un string, una lista de strings ([`'accuracy'`, `'mae'`, `'mse'`], por ejemplo), o mediante una o varias clases con `keras.metrics`.

- `history = mlp.fit(X, y, epochs=e, batch_size=bs, verbose=1, validation_split=vs)`: Método que realizará el entrenamiento de la red neuronal con los la matriz de características `X` y el vector o matriz de etiquetas `y`. El método correrá una cantidad `e` de épocas dividiendo la matriz `X` en segmentos o batches de tamaño `bs` filas. El parámetro `verbose` se utiliza para leer los resultados parciales de entrenamiento a medida que éste se realiza, con valor 1, o 0 para no mostrar nada. El parámetro `validation_split` reservará una proporción `vs` de los datos para hacer las pruebas en validación. El método arroja una historia en la variable `history` con los valores de función de pérdida y métricas para los sets de entrenamiento y validación en cada época.

- `test_results = model.evaluate(X, y, verbose=1)`: Este método ejecutará una prueba sobre la matriz de características `X` con el vector o matriz de etiquetas `y`. Se aplicarán las métricas y pérdidas especificadas en `model.compile()`.

- `y_cat = to_categorical(y, num_classes=nc)`: Transformará un vector de enteros `y`, de largo l en una matriz `y_cat` de dimensiones nc. Cada fila de `y_cat` será una representación de codificación *one hot* del elemento correspondiente de `y`. Útil para transformar etiquetas en problemas de clasificación, o para codificaciones de entrada.

1.3 Pytorch

Junto con TensorFlow, Pytorch es una de las librerías de aprendizaje automático de código abierto más utilizadas en aplicaciones e investigaciones de alto nivel. Mientras que TensorFlow ha sido desarrollado y es mantenido por el laboratorio Google Brain, Pytorch está basado en Torch, y actualmente es mantenido por su desarrollador, el laboratorio FAIR (Facebook's AI Research lab).

A diferencia de Keras, Pytorch tiene una curva de aprendizaje un poco más alta. Sin embargo, permite un desarrollo y control mucho más fino de las variables involucradas. En este caso es necesario desarrollar un método o función de entrenamiento, dependiendo de cómo se desee trabajar.

Dentro de las necesidades de esta tarea trataremos de mantener un nivel de complejidad similar al de Keras. Se ofrece el uso de esta librería como alternativa a Keras por si desea tomar ramos más avanzados en esta línea. Algunas clases y funciones se pueden importar de la siguiente manera:

```
import torch
from torch import Tensor
from torch.nn import Sequential, Linear, LazyLinear
from torch.nn import ELU, ReLU, Tanh, Sigmoid, Softmax
from torch.nn import CrossEntropyLoss, MSELoss
from torch.optim import Adam, SGD
from torchvision import transforms as tfs
from torch.utils.data import Dataset, DataLoader
```

Veamos cómo funcionan estas clases:

- `X = Tensor(x)`: Transforma la lista o arreglo NumPy `x` en un tensor `X`. Un tensor es una matriz de números con una dimensionalidad superior a 2. Se trata de la clase básica con la que Pytorch trabaja numéricamente.
- `X.shape`: Devuelve la forma del tensor `X`.
- `X = X.cuda()` o `X = X.cpu()`: Copia el tensor `X` a la RAM de la GPU o de vuelta a la RAM de la CPU. El computador debe tener una GPU con CUDA instalado, cosa que se puede verificar con el comando `torch.cuda.is_available()`.
- `x = X.numpy()`: Transforma el tensor `X` en un arreglo `x` de numpy.
- `model = Sequential(*args)`: Genera un contenedor secuencial para el modelo o red neuronal `model`. Se puede inicializar de dos formas, bien sea definiendo `*args` como varios parámetros separados por comas, o definiendo un diccionario, con cada llave o *key* como un string, y cada elemento como una tupla, bajo `OrderedDict`. A diferencia de la clase equivalente de Keras no existe un método `add`. De similar manera que con su equivalente, el órden importa.
- `model = User_class(*args)`: Alternativamente es posible definir una clase que herede de `torch.nn.Module`. Esta clase puede tomar tantas capas como se desee en el método `__init__()`, y deberá especificarse un método con nombre `foward()` donde se realizarán los cálculos. Se pueden agregar los métodos `fit()` y `evaluate()` para entrenar y evaluar la red según sus necesidades. El uso de una clase es uno avanzado que solo se recomienda para quienes quieran indagar más. Se requiere investigación.
- `model = model.cuda()` o `model = model.cpu()`: Copia la instancia de clase `model` a la RAM de la GPU o de vuelta a la RAM de la CPU. El computador debe tener una GPU con CUDA instalado, cosa que se puede verificar con el comando `torch.cuda.is_available()`.
- `model.train()`: Método que deja a la instancia de clase `model` en modo de entrenamiento. Permite la propagación de gradiente y modificación de los pesos de la red, y activa otras funciones deseables en entrenamiento.
- `model.eval()`: Método que deja a la instancia de clase `model` en modo de evaluación. Desactiva la propagación de gradiente y modificación de los pesos de la red, y desactiva otras funciones no deseables en evaluación.
- `layer = Linear(input, output, bias=True)`: Única clase de capa que necesitaremos. Se deben especificar en todo momento la cantidad de características de entrada, `input`, y las neuronas de la capa, `output`. El parámetro `bias` permite o no, la adición de un sesgo fijo a cada neurona, resultando en una transformación de datos de la forma $\hat{y} = A^t X + b$, con A una matriz de forma $input \times output$ y b un vector de largo $output$. No se aplica ninguna función de activación. Alternativamente se puede utilizar `LazyLinear` en la que el tamaño de la entrada se infiere.

- `activfn = ELU()`,`activfn = ReLU()`,`activfn = Tanh()`,`activfn = Sigmoid()`, o `activfn = Softmax()`: Define una función de activación para aplicar luego de una capa lineal o de otro tipo. Se debe agregar inmediatamente después de la capa a la que se desee aplicar en la llamada de `model = Sequential(*args)`.

- `summary(model, (bs, X.shape[1]))`: Una función equivalente al método de Keras `Summary` se puede instalar con `pip install torch-summary`. Se importa con `from torchsummary import summary`, y al llamarla se le debe pasar tanto la instancia de la red neuronal, como la forma del tensor de entrada en una tupla con el tamaño de batch deseado y el número de características de la matriz de entrada.

- `optimizer = SGD(model.parameters(), lr=l)` o `optimizer = Adam(model.parameters(), lr=l)`: Instancia un optimizador para la actualización de los pesos de la red. En cada caso se le deben pasar los parámetros de la red, y el radio de aprendizaje `lr`.

- `lossfn = CrossEntropyLoss()` o `lossfn = MSELoss()`: Define una función de pérdida para ser utilizada durante entrenamiento. Es posible definir una función propia que combine o utilice varias instancias de estas clases, en caso de ser necesario agregar tareas auxiliares o regulizadores.

- `tfs`: Es una sublibrería que permite agregar transformaciones para operaciones de preprocesamiento. Un par de ejemplos serían `tfs.ToTensor()` y `tfs.Normalize()`. Se puede utilizar `trans = tfs.Compose()` para componer una lista de transformaciones. Así, por ejemplo el arreglo `x` puede ser transformado en tensor y normalizado en un paso al hacer `X = trans(x)`.

- `Dataset`: Al crear una clase que herede de `Dataset` se puede controlar la carga de datos, bien sea desde archivos o directamente desde la RAM. Se debe especificar un método `__getitem__(self, idx)` que entrege los valores del batch a partir del índice `idx`. También es necesario especificar un método `__len__(self)` que determine el tamaño del set de datos.

- `loader = Dataloader(data, batch_size=bs, shuffle=True)`: Es una clase que se encargará de levantar los datos contenidos en la instancia `data` de la clase `Dataset` personalizada. Permite cortar los datos en un tamaño de batch `bs`. Si el parámetro `shuffle` se inicializa en `True`, entonces tomará porciones en un órden aleatorio de los datos (útil para no hacer *overfitting* duante entrenamiento).

Para cargar datos de una instancia `loader` e iterar por todo el set de datos completo conviene iterar:

```
for (labels, inputs) in loader:
...
```

De lo contrario se debe encontrar una forma de realizar un sampling de datos de los tensores para formar un batch. Una manera de hacerlo es utilizar la función `train_test_split` de `sklearn`, pero debe asegurarse de hacer un sampling con todos los datos de su matriz de entrenamiento. Otra forma es definir una función equivalente a un pop de filas con `np.vstack`, y llevar un registro de los índices de las filas de la matriz `X` que han sido utilizados durante la época, para luego transformarlos en tensor y convertirlos en una matriz de `inputs`; y lo mismo para la matriz o vector `y` y su equivalente de `labels`. Finalmente puede hacer uso de la herramienta `torch.utils.data.Subset`.

Dados dos tensores de datos `labels` e `inputs` el ciclo de cálculo de predicción y propagación de gradiente se declara así:

```
prediction = model(input)
#Se calcula una prediccion en el tensor de salida prediction
optimizer.zero_grad()
#El optimizador establece su gradiente en cero
loss = lossfn(prediction, labels)
```

```
#Se calcula la perdida
loss.backward()
optimizer.step()
#Se calculan los gradientes y se aplica un paso de actualizacion de pesos
mean_loss += loss.item()
#En el caso de acumular perdidas para graficos estas se suman para
#luego dividirse por el tamanho del set de datos
pred = prediction.argmax(dim=1, keepdim=True)
#En el caso de un problema de clasificacion es comun utilizar argmax
#para encontrar la prediccion de la red, dado que el vector de salida
#tendra una codificacion basada en varias clases.
```

1.4 Gym

OpenAI es una empresa líder en avances de inteligencia artificial fundada por Elon Musk. Uno de sus desarrollos más conocidos es el modelo de lenguaje GPT3, del que se ha hablado mucho por su impresionante cantidad de parámetros. Sin embargo, a nosotros nos interesa un desarrollo más sencillo de esta compañía. Se trata de *Gym*, un *toolkit* para trabajar con ambientes de laboratorio para aprendizaje reforzado. Su sitio web oficial se puede encontrar en `https://gym.openai.com/`.

Con *Gym* podemos simular un ambiente interactivo compatible con Google Colab. A diferencia del paradigma del aprendizaje supervisado, el aprendizaje reforzado requiere de un acceso interactivo con el ambiente para que el agente pueda aprender de él.

Los siguientes son algunos métodos y clases disponibles para trabajar con *Gym*:

- `env = gym.make(name)`: Clase que contiene la instancia del ambiente con nombre `'name'`.
- `env.render()`: Método que da inicio al renderizado de los episodios de la clase.
- `obs = env.reset()`: Método que resetea al ambiente al inicio de un episodio. Devuelve una observación del estado.
- `env.action_space`: Propidad de la clase que entrega información sobre el espacio de acciones posibles.
- `env.action_space.n`: Propiedad de la clase que entrega información sobre la dimensionalidad del espacio de acciones posible.
- `action = env.action_sapce.sample()`: Método que devuelve una acción aleatoria del espacio de acciones.
- `observation, reward, done, info = env.step(action)`: Método que realiza un paso de interacción de simulación con el ambiente. Devuelve una observación del estado, un valor de recompensa, un booleano que indica si el episodio ha teminado e información referente al debugueo del ambiente.
- `env.close()` Termina el renderizado del ambiente.
- `average_episodic_return(env, agent, episodes=10, max_steps_per_episode=500)`: Permite evaluar el retorno promedio que obtiene el agente `agent` al ejecutar su política sobre el ambiente `env` en tantos episodios como `episodes` cada uno de los cuales no puede superar un máximo `max_steps_per_episode` de pasos.
- `animate_agent(env, agent, max_steps = 400)`: Permite generar una animación del comportamiento del ambiente `env` bajo la política que entregue el agente `agent` en un máximo de pasos `max_steps`.

Advertencia. Es importante considerar un bug asociado al ambiente Gym. Si se interrumpe una función o método que mantenga al ambiente en constante simulación es posible que toda la sesión de Google Colab colapse. Si esto sucede existe la posibilidad de perder las variables almacenadas en RAM. Para evitar que esto suceda puede trate de no interrumpir las celdas que estén en simulación del ambiente. Si se ve en la necesidad de realizar interrupciones de ese tipo se le recomienda guardar variables importantes (como la matriz Q) en la memoria de disco o en su drive. También puede incorporar bloques del tipo `try`/`except`, en las que maneje las excepciones con el uso de `env.close()`.

1.4.1 Pendulum-v0

El ambiente `Pendulum-v0` simula un péndulo que cuelga sobre un eje. A este péndulo se le puede aplicar un torque continuo para tratar de girarlo y llevarlo a una posición vertical. El objetivo es mantener el péndulo en esa posición la mayor parte del tiempo posible.

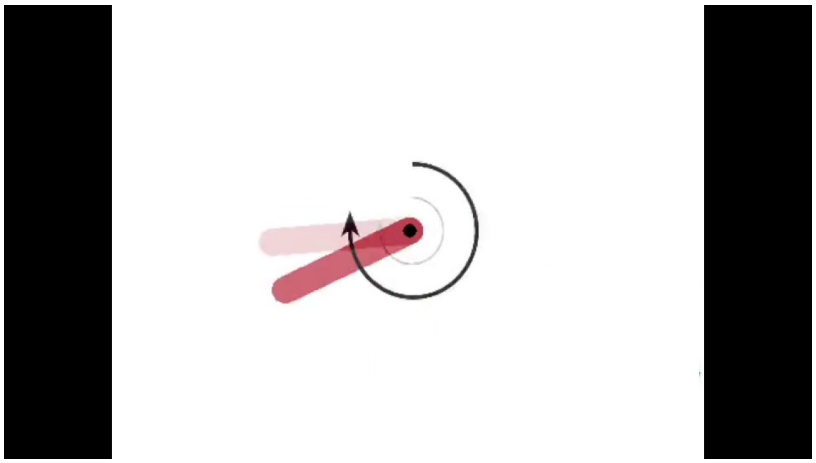

Figure 1: Ambiente Pendulum-v0 utilizado en la tarea.

El ambiente entrega una observaciónn del ángulo del péndulo y su velocidad angular. Esta observación se presenta como una lista de tres elementos continuos según se muestra en la Tabla 1:

Indice	Observación	Min	Max
0	$cos(\theta)$	-1	1
1	$sen(\theta)$	-1	1
2	$\dot{\theta}$	-8	8

Table 1: Sensores involucrados en la observación

Al ser de naturaleza continua, los valores deben ser discretizados para poder trabajar con las técnicas de aprendizaje reforzado vistas en clases. Para realizar una discretización se le sugiere estudiar los rangos en los que caen las variables, y establecer una serie de intervalos para cada variable, y numerarlos desde cero hasta $n-1$, con n el número de intervalos para esa variable. Cuando un valor observado cae en un intervalo numerado esa observación pasa a tener como discretización el valor del entero que representa al intervalo. En el caso de múltiples variables, se debe establecer una codificación para rescatar un único número que represente la discretización.

De similar manera el actuador generará un torque continuo sobre el eje de rotación como se ve en la Tabla 2.

Indice	Acción	Min	Max
0	τ	-2	2

Table 2: Acción de torque

El estado inicial en el que se encuentra el ambiente luego del uso del método `reset` se presenta como una variable aleatoria que puede tomar valores según las siguientes distribuciones:

$$\theta \sim U(-\pi, \pi)$$
$$\dot{\theta} \sim U(-1, 1)$$

Finalmente, existe una función de recompensas específica para este ambiente cuya ecuación, en cada paso del episodio, viene dada por:

$$R(\theta, \dot{\theta}, accion) = -(\theta^2 + 0.1\dot{\theta} + 0.001accion^2) \tag{1}$$

Como θ está normalizado entre $(-\pi, \pi)$, el valor de la recompensa en cada paso varía entre $(-16.2736044, 0)$.

2 Métricas de rendimiento

Para conocer el rendimiento de un modelo de ANN ya entrenado, es necesario probar su capacidad para predecir nuevos datos (ie. datos no usados en su entrenamiento). Ya que la cantidad de datos de la que se dispone es limitada, es necesario separar de antemano una porción de los datos (set de test) que se usará para obtener nuestras métricas.

En el caso del aprendizaje reforzado nos valdremos de la recompensa recibida por el agente a lo largo de su entrenamiento, y la recompensa promedio por episodio una vez éste esté entrenado.

2.1 Métrica de evaluación de un problema de regresión

Se utilizará el error cuadrático medio (mean square error), tanto como métrica como función de pérdida. Se hace esto, pues a diferencia de un problema de clasificación, aquí se puede utilizar (en un problema de calsificación la precisión es no lineal y por lo tanto no sirve como pérdida), y para todo problema de optimización es deseable contar con la variable con la que se está optimizando. Además, al ser una función convexa tiene propiedades que la hacen deseable como función de pérdida.

Por otro lado, trabajaremos con un problema de regresión, es decir, trataremos de predecir un valor real con la mayor precisiónposible. Una de las métricas más simples para esta tarea es el puntaje de error absoluto medio (mean absolute error), que nos permitirá ver qué tan lejos está nuestra predicción del modelo para un valor de *ground truth*, obtenido del set de test. La ventaja que tiene esta métrica sobre la anteriore es que es mucho más interpretable, en especial por estar en las mismas unidades que nuestra variable a optimizar.

2.2 Métrica de evaluación de un problema de aprendizaje reforzado

En un problema de aprendizaje por refuerzo el agente tiene como objetivo maximizar su recompensa. Por un lado, la remuneración puede depender de múltiples factores que no generalizan a todas las situaciones. Una metodología relativamente universal, que no depende mucho de estos factores, es la medición de la recompensa a lo largo de cada episodio. Esta recompensa se puede medir, sumando todas las gratificaciones recibidas a lo largo de cada episodio, y graficando su evolución a lo largo del entrenamiento. Otra forma de medir el progreso del agente es calcular la recompensa media cada tantos episodios de entrenamiento. Esto se consigue pausando el aprendizaje, y utilizando el ambiente para evaluar, en n episodios de test, un promedio de la retribución recibida.
Si se toma nota de ambos procedimientos, el primero es un equivalente a revisar el progreso de la métrica con los datos de entrenamiento, mientras que el segundo la estudia con datos equivalentes a test.

3 Actividades

3.1 Problema de regresión

En un problema de regresión el objetivo es tratar de predecir un valor numérico específico, bien sea un entero o un valor en punto flotante. En este caso vamos a tratar de predecir el precio de un diamante dadas varias características. Utilizaremos redes neuronales lineales, conocidas como perceptrones multicapa, como estrategia inductiva.

Prepare un *jupyter notebook* con las respuestas. Descargue el set de datos utilizando la siguiente dirección de *google drive* https://drive.google.com/file/d/15Z0tkJW6NTdvi9q0G3vQDgRyniXzqEzG. Puede hacer uso de la instrucción *gdown –id [id]* donde el [id] se reemplaza por lo que está marcado en rojo. Esta instrucción importa los datos a la carpeta en la que se encuentra el archivo *jupyter*.

Este set de datos contiene varias series de datos de propiedades útiles para la tasación de un diamante. Algunas de estas propiedades tienen un espacio de medición en punto flotante. Estas series corresponden a las columnas con nombres: **carat**, **x**, **y**, **z**. Donde **carat** hace referencia al peso del diamante, su quilate, medido en gramos; mientras que los valores hacen referencia a sus medidas básicas en milímetros. Otros valores, son el **depth**, altura de un diamante, medida desde el culet (punta inferior del diamante) hasta la tabla, dividida por el diámetro promedio de la faja, $2 * z/(x + y)$, en porcentaje, y el **table** (tabla, faceta superior más grande), que es el ancho de la tabla del diamante expresado como porcentaje de su diámetro medio.

Otras propiedades tienen un espacio de medición discreto, pero no vienen expresadas en números enteros, y son muy importantes para la tasación. Estas son la calidad del corte (**cut**), graduada en cinco categorías: Fair, Good, Very Good, Premium, Ideal; el **color**, con categorías en letras desde la peor, D, hasta la mejor, J; y finalmente la claridad (**clarity**) medido también con un sistema de 8 categorías: I1 (peor), SI2, SI1, VS2, VS1, VVS2, VVS1, IF (mejor). Ignorar estas propiedades puede ser perjudicial para una buena regresión.

Como podrá deducir, el objetivo de esta actualidad se encuentra en la columna de precios (**price**). Los valores se encuentran en dólares, y los utilizaremos como nuestro vector objetivo.

Al igual que en la tarea anterior, la columna **Usage** servirá para separar los sets para entrenamiento, validación y test.

1. Haga una buena lectura de los datos contenidos en el archivo csv. Para ello identifique y transforme los datos a un formato adecuado para su trabajo en esta actividad. Proponga, explique y justifique una estrategia para la limpieza y transformación de los datos. Mencione cómo procederá con cada columna, o grupos de columnas, no deje ninguna fuera. Ponga especial énfasis en los datos no numéricos (Hint: haga uso de las funciones de Pandas introducidas aquí para idear una forma de convertir los datos no numéricos a alguna codificación útil). Construya la matriz de características X, y el vector de objetivos y, recuerde que debe utilizar la columna **Usage** para separar previamente un set de test. Utilice el atributo `shape` para mostrar la forma de cada matriz.

2. Seleccione por lo menos una estrategia de preprocesamiento de datos y aplíquela. Describa en sus propias palabras de qué se trata esa estrategia, y la razón de porqué la seleccionó, o de porqué espera con ella una buena predicción. Divida el set de datos con valores originales "ToV" en entrenamiento y validación. Si utiliza la librería `keras` puede omitir este paso, la propia librería se encarga de hacer la separación, pero se le debe declarar (ver arriba). En el caso de `pytorch` puede recurrir a la función `train_test_split` de `sklearn` o definir una propia. Mencione cuál es su elección de separación de los sets (porcentajes o razones), justifique dicha elección, y utilice el atributo `shape` para mostrar la forma de cada matriz disponible.

3. Construya un perceptrón multicapa. Seleccione entre `keras` o `pytorch` como librería para su red neuronal. En esta etapa el perceptrón solo puede tener una capa oculta, además de sus capas de entrada

y de salida. Defina la cantidad de neuronas por capa y las funciones de activación. Utilice `mse` como función de pérdida, y tanto `mae` como `mse` como métricas de rendimiento. Entrene su red durante tantas épocas como le parezca adecuado, utilizando un bath de 32. Haga un estudio de sensibilidad de parámetros. Proponga hasta 5 pares de valores de cantidades de neuronas para las capas inicial y oculta, y utilice estas siguientes tres funciones de activación: `relu`, `tanh` y `sigmoid`. Muestre dos tablas con los valores de `mse` y `mae` en cada una (según lo especificado dos tablas de 5x3). Seleccione el modelo con los mejores valores de métricas y realice una prueba en el set de test. Despliegue un gráfico de evolución de ambas métricas en el tiempo. ¿Cómo se correlacionan, con cada métrica, los resultados en los sets de entrenamiento y validación? Esta pregunta requiere un valor máximo de `mae` de 800 en el mejor modelo para ser considerada correcta.

4. En esta actividad utilizaremos aprendizaje profundo en una versión simple. Seleccione los hiperparámetros del mejor modelo del problema anterior: par (neuronas de capa de entrada, neuronas de capa intermedia) y función de activación. Construya el modelo, pero antes de compilar o entrenar con él deberá agregar capas ocultas intermedias (normalmente entre la de entrada y la intermedia ya definida, pero esto no es obligatorio). Realice otro estudio, esta vez con una tabla de hiperparámetros que considere el número de capas ocultas agregadas, y la cantidad de neuronas por capa agregada. Seleccione el modelo con los mejores valores de métricas y realice una prueba en el set de test. Despliegue un gráfico de evolución de ambas métricas en el tiempo. Despliegue la nube de puntos de los valores predichos y los valores reales de cada diamante (gráfico precio vs precio). Haga una gráfica comparativa entre los resultados del modelo de la pregunta 3 y el mejor modelo aquí seleccionado (dos gráficos precio vs precio, uno con el modelo de esta pregunta y otro con el modelo de la pregunta anterior). En cada caso haga una regresión lineal simple y compare los valores R^2 de cada predicción. Presente una explicación simple de porqué el mejor modelo supera al otro. Esta pregunta requiere un valor máximo de `mae` de 600 para ser considerada correcta.

3.2 Problema de aprendizaje reforzado

Entre los diferentes paradigmas de aprendizaje hemos visto los dos problemas más comunes para el aprendizaje supervisado: clasificación y regresión. También existen otros paradigmas de aprendizaje que hacen referencia al no uso de las etiquetas, como el aprendizaje no supervisado o semi-supervisado. Finalmente tenemos el aprendizaje reforzado.

En esta oportunidad utilizaremos el *toolkit Gym* para simular un ambiente de una nave espacial aterrizando en la luna (ver figura 2). El trabajo interactivo de estos ambientes nos permite aplicar una serie de técnicas, dentro de las cuales *q-learning* es una de las más adecuadas, de las vistas en clases para este ambiente. Sin embargo esta técnica requiere que el ambiente tenga un espacio de estados discreto, característica que no se cumple para todas las variables de estado.

5. Inicialice *Gym* con el ambiente `Pendulum-v0` utilizando los códigos entregados en el template de la tarea. Utilice las funciones provistas por el *toolkit* para estudiar tanto el espacio de acciones como el espacio de observaciones disponibles por el ambiente. Verifique que correspondan con las características señaladas más arriba. Corra varios episodios independientes para poder extraer suficientes datos de los sensores (observaciones). Despliegue un histograma de cada sensor según las observaciones realizadas. Defina una función de nombre `state_for` que genere una discretización de las variables de la observación, utilice la información de los histogramas como apoyo para esta tarea. Esta función debe recibir todas las variables de los sensores que entrega el ambiente haciendo uso del método `step`, y transformarla en un único entero. De la misma manera construya una función `action_for` que discretice las acciones. Esta función debe tomar un índice en un rango definido por usted, y ejecutar una acción en un rango compatible al actuador (ver Tabla 2).

   Hint: Para no generar una matriz Q demasiado grande, la cantidad de posiciones en la matriz, es decir la cantidad de índices de estado multiplicada por la cantidad de índices de acciones no debe ser muy grande. Se recomienda que dicha multiplicación sea cercana a 1000, para tener un buen balance entre resolución y tamaño de espacio de Q a visitar.

6. Construya una función que represente un agente. Esta función debe tomar un valor de observación, convertirlo en un valor de estado utilizando la función `state_for` definida en la pregunta anterior, y acceder a la matriz Q con ese estado. También debe poder convertir, un índice en una acción mediante el uso de la fución `action_for` para acceder a ella. Es decir, debe acceder a $Q(s, a)$, para actualizar su valor (decidir una acción $\text{argmin}_a Q(s, a)$ en dicho estado y devolver el valor Q de esa acción). Desarrolle primero un agente que para cualquier estado ejecute una acción aleatoria, despliegue la animación correspondiente con la función `animate_agent`, y el valor resultante de `average_episodic_return`. Entrene el agente definido arriba con *q-learning*. Utilice una cantidad de episodios apropiada al tamaño de la matriz Q. Puede utilizar una política de exploración ϵ-greedy, o una función a tramos que tome probabilidad de exploración ϵ según el episodio en el que se encuentre [1]. Despliegue nuevamente la animación de `animate_agent` y el valor de `average_episodic_return`. Presente una gráfica de la evolución de las recompensas totales, y de las recompensas promedios recibidas por episodio. Queda completamente prohibido el uso de *DeepQ*.

7. (BONO): Investigue sobre el algoritmo de aprendizaje reforzado profundo *DeepQ*. Explique sus principales características y construya un modelo en el framework de redes neuronales utilizado en la pregunta 3, no lo entrene. ¿Cómo reemplazaría la función de pérdida para la propagación de gradiente? Redacte el código, no es necesario probarlo ni debuguearlo, solo debe estar conceptualmente bien. En el caso de Keras suponga que puede puede definir una función de nombre `deep_q_loss` con las entradas y salidas correspondientes definidas por usted. Si bien no es necesario agregar la memoria de *replay*, sí es indispensable mencionar su uso y funcionamiento a nivel teórico. Esta pregunta es opcional, y de realizarla su puntaje reemplaza al puntaje de la peor pregunta obtenida hasta ahora, siempre y cuando supere dicho puntaje. En caso contrario no se considera. Recuerde citar sus referencias.

El entregable es un archivo *jupyter notebook* (de extensión ".ipynb") con su nombre y apellido, número de alumno, y el desarrollo de la actividad. Cerciórese de que la opción "Omitir el resultado de las celdas al guardar este notebook" está deshabilitada, en este caso no es necesario hacer uso de la aceleración de hardware o "GPU", pero podrá beneficiarle. Puede hacer esto en el menú Edición⇒ Configuración del notebook. No se corregirán los notebooks que no tengan los resultados disponibles.

References

[1] S. Gadgil, Y. Xin, C. Xu, *Solving The Lunar Lander Problem under Uncertainty using Reinforcement Learning*, arXiv preprint arXiv:2011.11850, 2020

[2] Mr Ko, `https://ai-mrkogao.github.io/openai/pendulum/`, 23 de Junio de 2021, 21:17 hrs