Universidad Autónoma Metropolitana

Unidad Cuajimalpa

Práctica 5

Aprendizaje Supervisado para diseñar un coche autónomo

Integrantes:

→ Barajas Sánchez Maximiliano 221302628

→ Victoria Nava Natalia 2193034858

Materia: Neurociencias Computacionales y aprendizaje automático

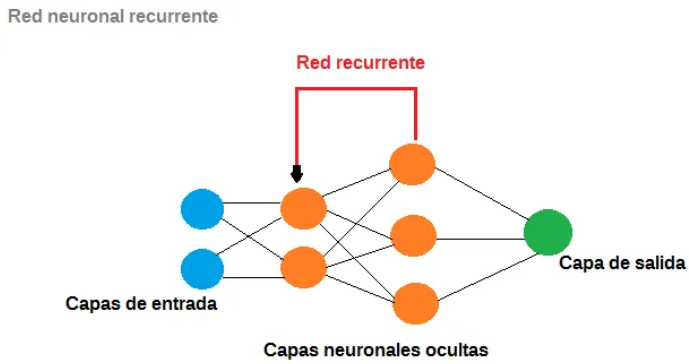
Introducción

A lo largo de esta práctica aplicaremos el conocimiento adquirido anteriormente sobre el perceptrón multicapa con especial énfasis en la regresión, esto se llevará a cabo entrenando una red para el manejo de un auto en el simulador de licencia libre TORCS.

Marco teórico

Perceptrón multicapa

Las redes neuronales se componen de diversas capas, reciben entradas u observaciones como también se acostumbra a nombrarles y regresan una salida, en nuestro caso al tratarse de una regresión será una o múltiples salidas continuas.



Las redes neuronales tradicionales en general poseen 3 tipos de capas:

- **Capa de entrada:** Conjunto de neuronas que se encargan de recibir las entradas.
- **Capas ocultas:** Estas capas se encargan a grandes rasgos de procesar las entradas entre sí.
- **Capas de salida:** Neuronas que se encargan de manifestar una salida en torno a los resultados de las capas ocultas.

Para este tipo de red neuronal es muy importante el conjunto de datos. Que se divide en 2 partes:

- **Muestra de entrenamiento:** Corresponde a los registros de datos para entrenar la red neuronal, cierto porcentaje del conjunto de datos debe asignarse a la muestra de entrenamiento.
- **Muestra de prueba:** Es un conjunto independiente de datos que se utilizan para dar seguimiento de los errores durante el entrenamiento.

Herramientas utilizadas a lo largo de la práctica.

A lo largo de la práctica se hizo uso de dos herramientas principales, todo se implementó en Python debido a la existencia de la biblioteca Sci-Kit Learn dado que de tal manera no debíamos implementar nuestra propia red neuronal, por otra parte se usó de igual forma la biblioteca pickle de tal manera que pudiéramos guardar la red neuronal ya entrenada y almacenarla en forma de un archivo con terminación .drv, todo se realizó en el simulador de carreras TORCS, cabe recalcar que realizamos modificaciones pertinentes al archivo drivers en la clase neuro controlador, dichos cambios a continuación:

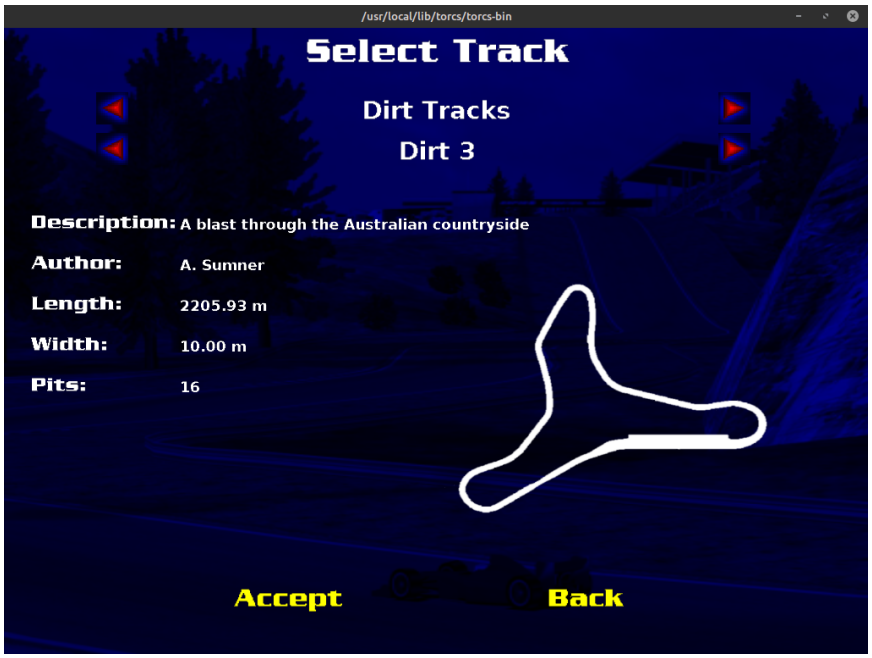
```
    # Una vez que ya tienen su red, este método se invocará de manera automática
    # cada vez que el servidor mande información de los sensores (el entorno del auto).
    # Como salida deben regresar los valores de los actuadores del auto.
    def drive(self, sensors):
        # 1. Tomar la información de los sensores necesarios y ponerlos en un vector RENGLON X como lo
        #    espera la red neuronal.
        # Lista total de nombres del diccionario de sensores.
        listaSensores = ['angle', 'trackPos']
        # Como cada elemento del diccionario es una lista, si el valor es un
        # solo elemento debemo usar el índice 0.
        # Ejemplo: sensors['angle'][0]
        lista =[]
        for clave in listaSensores:
            lista.append(sensors[clave][0])
        trackSensors=sensors['track']
        lista.extend(trackSensors)
        # Despues de extraer los valores y ponerlos en lista.
        X = np.array(lista, dtype=float)
        # 2. Usar predict de nuestra red con el vector X y recibir la predicción y (será un vector).
        y = self.neurocontrolador.predict([X])[0]
        #3. Poner los valores de la salida y en las entradas del diccionario de abajo y regresarlo.
        #   Debemos normalizar los valores de salida para que estén en su intervalo correcto:
        #   acelerador [0,1], freno [0,1],velocidad caja [-1,0,1,...,6], volante [-1,1] y .
        #Verificamos que el acelerador predicho por la red se encuentra en el intervalo [0,1] y de no ser así le asignamos un valor
        if (y[0]>1): y[0] =1 #Asignamos 1 si se pasa del intervalo
        if (y[0]<0): y[0] =0 #Asignamos 0 si esta por debajo del intervalo
        #Repetimos un proceso indistino para el freno
        if (y[1]>1): y[1] =1
        if (y[1]<0): y[1] =0
        #Para el caso de la caja de cambions, redondeamos el valor y aplicamos algo similar
        y[2] = round(y[2])
        if (y[2]>6): y[2]=6 #Asignamos 6 de ser que se pase lo predicho por la red
        if (y[2]<-1): y[2] = -1 #Asignamos -1 de ser que este por debajo
        #Procedemos a normalizar la prediccion del volante
        if (y[3] >1.0): y[3] = 1.0 #Asignamos 1 en caso de pasarse
        if (y[3]<-1.0): y[3] = -1.0 #Asignamos -1 en caso de faltarle
        print("Predicciones de la red",y)
        #Asignamos a cada llave del diccionario el valor que espera de entrada el juego
        return {'accel' : y[0],'brake' : y[1],'steer' : y[3],'gear'  : y[2],'clutch':0,'focus' :0.0,'meta'  :0}

# SimpleExponentialFunctionDriver
```

Notemos que decidimos darle a la red total control del auto y normalizamos cada valor al intervalo correcto, tomamos la convención de que si se pasaba del intervalo asignamos el valor más grande del mismo y si estaba por debajo del intervalo el más pequeño, en el caso de la caja antes de hacer eso redondeamos el valor arrojado por la red neuronal.

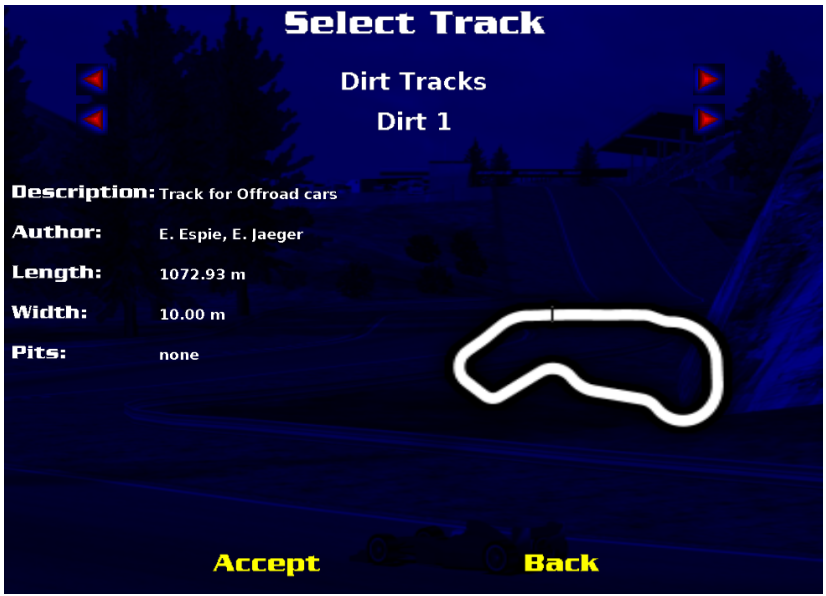
Desarrollo:

Como primer circuito decidimos utilizar el siguiente:



Decidimos utilizar dicho circuito como el primero dado que consideramos era el más completo de los de tierra y para hacer pruebas en especial para el volante fue de gran ayuda, se generó un archivo de salida como se estableció en la presentación el cual contenía los datos de 2 vueltas de este circuito, dicho archivo se guardó (para más tarde entrenar la red neuronal) con ayuda de un control de xbox one, tuvimos que hacer de igual manera algunos ajustes al archivo drivers en la sección del controlador por gamepad dado que al utilizar el gatillo como acelerador marcaba valores no válidos en el juego y lo cerraba.

Como segundo circuito decidimos utilizar uno menos demandante que es del apartado de los circuitos con tierra, elegimos el circuito a continuación:



Manualmente juntamos los archivos salida.txt para entrenar a la red neuronal con los siguientes parámetros:

```python
import numpy as np
import pickle
from sklearn.neural_network import MLPRegressor
from sklearn.model_selection import train_test_split
#Leemos los datos con los cuales entrenaremos a la red
datos=np.loadtxt("./salida.txt")
#Utilizamos solo los datos necesarios
indice_auxiliar=6
numero_track=19
indice_final=indice_auxiliar+numero_track
idx=np.r_[0:2, indice_auxiliar:indice_final]
X=datos[:,idx]
y=datos[:,indice_final:]
datasets=train_test_split(X,y,test_size=0.2)
train_X, test_X, train_y, test_y=datasets
#Entrenamos el modelo
mlp=MLPRegressor(hidden_layer_sizes=(18,18,18),solver='adam',tol=1e-15,learning_rate='adaptive',max_iter=580000000,activation='identity')
mlp.fit(train_X,train_y)
y_pred=mlp.predict(test_X)
puntos=mlp.score(test_X,test_y)
print("\nEl modelo obtuvo los siguientes puntos: %f"%(puntos))
nombreArchivo='mlp_driver.drv'
#Guardamos la red ya entrenada
pickle.dump(mlp,open(nombreArchivo,'wb'))
```

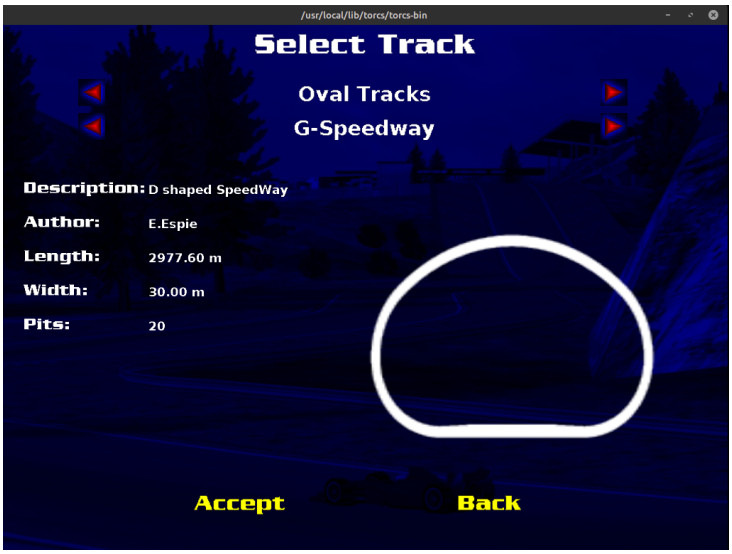
Para nuestra red decidimos tomar 3 capas de 18 neuronas cada una, el optimizador ADAM a recomendación de la bibliografía, un coeficiente de aprendizaje adaptativo y un máximo de iteraciones de 580000000 a raíz de una recomendación realizada por el profesor en la Unidad de Enseñanza y Aprendizaje de Inteligencia Artificial impartida anteriormente, de igual manera tomando en cuenta dicha UEA decidimos utilizar por función de activación la identidad dado que las otras funciones de activación resultaba en un manejo incorrecto de parte de la red en cuanto al volante, los vídeos de la red conduciendo estos dos circuitos se encuentran a continuación:

Circuito 1: https://www.youtube.com/watch?v=IdgvkHNZ-b4

Circuito 2: https://www.youtube.com/watch?v=Rtq4sBYTQqo

Entre las observaciones que pudimos realizar de este caso es que la red al igual que nosotros prefiere mantenerse en primera velocidad en los circuitos fangosos para evitar patinar sobre la pista y resulta adaptarse bastante bien a condiciones ya sean de fango o de calle dado que la probamos en otros circuitos similares sin haberla entrenado para ellos.

FInalmente a requisito de lo establecido en la práctica probamos la red con el siguiente circuito:



Este circuito lo elegimos como el tercero dado que pareciera ser más sencillo que los anteriores y efectivamente lo fue así, dado que la red neuronal logró completar una vuelta de el, el vídeo se encuentra en el link a continuación, https://youtu.be/eKZMxL7bkFQ?feature=shared , sin embargo lo realiza de una manera un tanto extraña y nuevamente manteniendo una velocidad baja en la caja de cambios a raíz de los constantes bajones de velocidad requeridos por los circuitos anteriores al ser circuitos de tierra, dado que en el conjunto de entrenamiento a raíz de nuestras habilidades no muy amplias en la conducción de autos virtuales resultó en un uso escaso de la caja de cambios aunado al hecho de que realmente necesitaba la red mantenerse controlada en los circuitos de tierra y consecuentemente evitaba aumentar demasiado la velocidad.

Conclusiones:

Podemos concluir que la recopilación de datos adecuados para entrenar la red es casí si no es que mas importante que la implementación de la misma, en repetidas ocasiones debido a utilizar inicialmente información de una sola vuelta resultaba que la red no realizaba ningún movimiento en el auto o presentaba otro tipo de problemas como detenerse totalmente justo antes de la meta, de igual manera la elección de una función de activación es vital, esto a raíz de que causo problemas a la hora de hacer uso de otras funciones de activación que no fueran la identidad, usamos la identida a raíz de una recomendación del profesor realizada en la siguiente clase videograbada de una UEA anterior: https://drive.google.com/file/d/1_Axvjr4wNzt5Ut9JRn5xIw2coqQp7ueG/view?usp=sharing

Referencias Bibliográficas:

- Checcucci, E., Autorino, R., Cacciamani, G. E., Amparore, D., De Cillis, S., Piana, A., ... & Porpiglia, F. (2019). Artificial intelligence and neural networks in urology: current clinical applications. *Minerva urologica e nefrologica= The Italian journal of urology and nephrology*, *72*(1), 49-57.
- Kramer, O., & Kramer, O. (2016). Scikit-learn. *Machine learning for evolution strategies*, 45-53.
- Torres, J. (2018). *DEEP LEARNING Introducci—n pr‡ ctica con Keras*. Lulu. com.
- LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *nature*, *521*(7553), 436-444.

Notas:

Los archivos para entrenar la red junto con la propia red ya entrenada se encuentran en la carpeta adjunta en formato zip a la entrega de la práctica, se realizaron múltiples redes a lo largo de la realización de la práctica, adjuntos se incluye la carpeta por nombre circuito_entregable.zip el cual contiene la red que probó ser la de mayor éxito y otro archivo adicional, circuito_entregable_2.zip . el cual contiene otra red neuronal entrenada con los mismos parámetros y datos la cual tiene un desempeño diferente.