Universidad Mariano Gálvez, Zacapa

Conoceréis la verdad y la verdad os hará libres… Juan 8:32

Ingeniero: Gregorio Henríquez

Inteligencia Artificial

2023

Proyecto Final

Bryan Emanuel Paz Ramírez - 1190-19-3929

Emmanuel Alexander Cabrera Álvarez – 1190-19-5728

IX semestre

Bot de Discord

Python

Python es un lenguaje de alto nivel de programación interpretado cuya filosofía hace hincapié en la legibilidad de su código, se utiliza para desarrollar aplicaciones de todo tipo, ejemplos: Instagram, Netflix, Spotify, Panda3D, entre otros. Se trata de un lenguaje de programación multiparadigma, ya que soporta parcialmente la orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, dinámico y multiplataforma. Administrado por Python Software Fundación, posee una licencia de código abierto, denominada Python Software Fundación License. Python se clasifica constantemente como uno de los lenguajes de programación más populares.

Python es el lenguaje elegido para la programación de nuestro Bot gracias a la librería de Discord.py que esta es una librería que facilita la creación de Bots para Discord.

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.1 Actualmente es utilizado tanto en la investigación como en los productos de Google:min 0:15/2:17 2:p.2 1:0:26/2:17 frecuentemente reemplazando el rol de su predecesor de código cerrado, DistBelief. TensorFlow fue originalmente desarrollado por el equipo de Google Brain para uso interno en Google antes de ser publicado bajo la licencia de código abierto Apache 2.0 el 9 de noviembre de 2015.

TensorFlow fue una de las librerías que utilizamos para la manipulación de redes neuronales, se utiliza junto con TensorFlow datasets que es un complemento de la librería de TensorFlow es una página que almacena distinta información en grandes cantidades para facilitarnos a la hora de la utilización de una Red Neuronal ya que con este existe la capacidad de que la red aprenda mucho más rápido conforme a los ejemplos que se le pasan.

Numpy

es una biblioteca para el lenguaje de programación Python que da soporte para crear vectores y matrices grandes multidimensionales, junto con una gran colección de funciones matemáticas de alto nivel para operar con ellas. El precursor de NumPy, Numeric, fue creado originalmente por Jim Hugunin con contribuciones de varios otros desarrolladores. En 2005, Travis Oliphant creó NumPy incorporando características de la competencia Numarray en Numeric, con amplias modificaciones. NumPy es un software de código abierto y cuenta con muchos colaboradores.

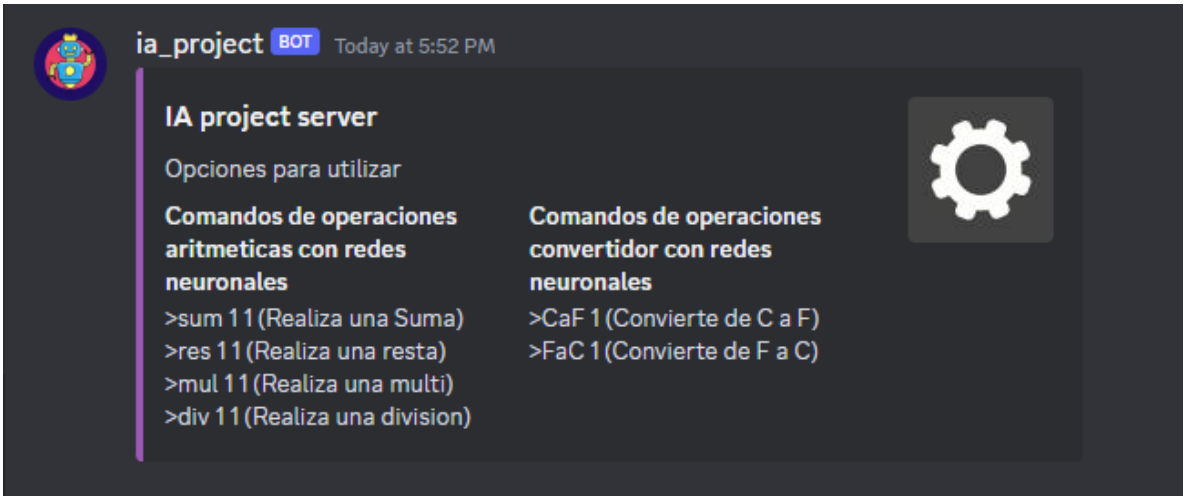
Numpy fue la librería que se utilizo para el manejo de datos gracias a los array de numpy gracias a la facilidad y rapidez que brindan estos array.

Matplotlib.pyplot

Es una librería que nos ayuda a graficar en tiempo real, como por ejemplo en nuestro proyecto nos dio la posibilidad de ver la manera en que este iba aprendiendo conforme los ejemplos que le dábamos como por ejemplo:

Proyectos

Presentamos el siguiente Bot en una plataforma que es explicita para hablar mientras se esté jugando, pero también existen diversos tipos de bots en esta aplicación de chat, audio y video, nuestro Bot tiene 6 funcionalidades básicas aplicadas donde antes de mostrar la funcionalidad se presentara el mensaje de información del Bot el cual es el siguiente:



Realiza Suma

Tenemos los datos de entrenamiento para la Red Neuronal

```
xt = np.array([[0.0, 0.0], [0.0, 1.0], [1.0, 0.0], [1.0, 1.0]])
yt = np.array([[0.0], [1.0], [1.0], [2.0]])
```

Tenemos 3 capas donde estas tienen el nombre de oculta1 esta contiene 3 neuronas y se le indica que el ingreso principal va a ser 2 datos, oculta2 esta de la misma manera se le indica que tendrá 3 neuronas y por último tenemos la capa de salida que este contara con una única neurona, todo esto agregándolo al modelo con el que se realizara el aprendizaje.

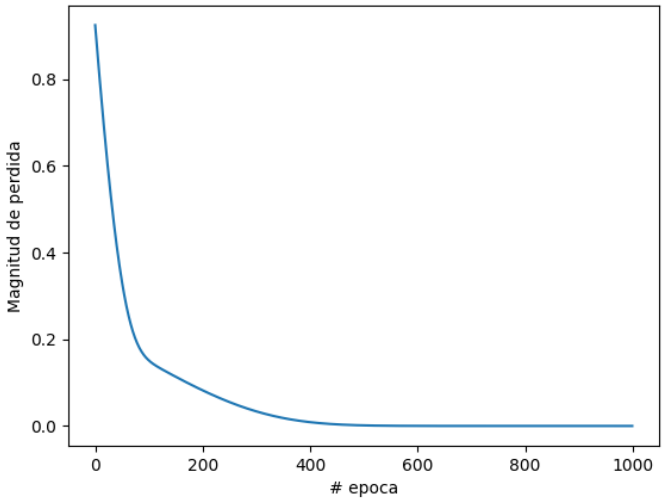
```
oculta1 = tf.keras.layers.Dense(units=3, input_shape=(2,))
oculta2 = tf.keras.layers.Dense(units=3)
salida = tf.keras.layers.Dense(units=1)
model = tf.keras.Sequential([oculta1, oculta2, salida])
```

Luego de tener el modelo listo lo compilamos con el optimizador Adam y le verificamos que el método de perdida sea mean_squared_error.

Seguido de enviamos el mensaje de que la red neuronal esta trabajando y empezamos el método de enseñanza para la red neuronal pasando por 1000 epocas con los datos de practica anteriormente ya mostrados.

```
model.compile(
    optimizer='adam',
    loss='mean_squared_error'
)

await ctx.send("La red neuronal esta procesando...")
historial = model.fit(xt, yt, epochs=1000, batch_size=4)
```

Presentamos la muestra de como es que va aprendiendo la red del bot en la siguiente imagen:



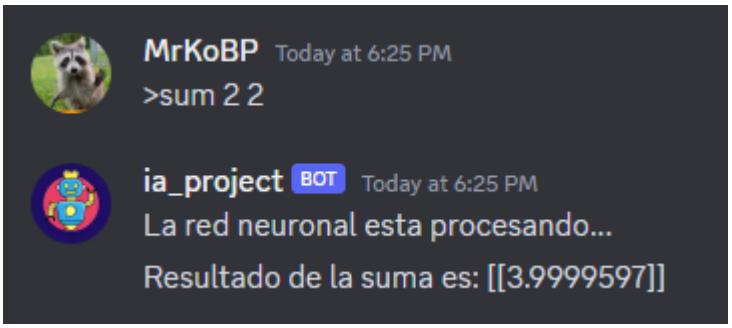
El Código del siguiente graficado es:

```
plt.xlabel("# epoca")
plt.ylabel("Magnitud de perdida")
plt.plot(historial.history['loss'])
plt.show()
```

Por último, realizamos la predicción y mostramos el resultado

```
prediccion = model.predict(xtest)
await ctx.send("Resultado de la suma es: " + str(prediccion))
```

Ahora pasamos al chat de Discord y mostramos como se ve el resultado:



Realiza Resta

Tenemos los datos de entrenamiento para la Red Neuronal

```
xt = np.array([[0.0, 0.0], [0.0, 1.0], [1.0, 0.0], [1.0, 1.0]])
yt = np.array([[0.0], [-1.0], [1.0], [0.0]])
```

Tenemos 3 capas donde estas tienen el nombre de oculta1 esta contiene 32 neuronas y se le indica que el ingreso principal va a ser 2 datos, oculta2 está de la misma manera se le indica que tendrá 32 neuronas y por último tenemos la capa de salida que este contara con una única neurona, todo esto agregándolo al modelo con el que se realizara el aprendizaje.

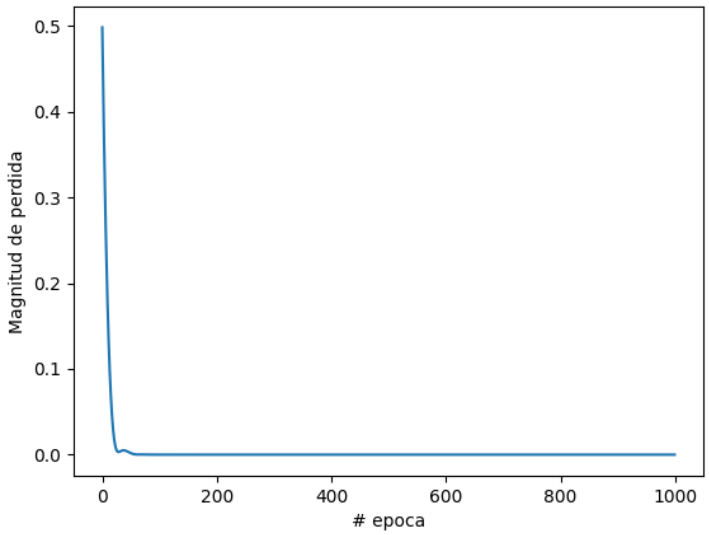
```
oculta1 = tf.keras.layers.Dense(units=32, input_shape=(2,))
oculta2 = tf.keras.layers.Dense(units=32)
salida = tf.keras.layers.Dense(units=1)
model = tf.keras.Sequential([oculta1, oculta2, salida])
```

Luego de tener el modelo listo lo compilamos con el optimizador Adam y le verificamos que el método de perdida sea mean_squared_error.

Seguido de enviamos el mensaje de que la red neuronal esta trabajando y empezamos el método de enseñanza para la red neuronal pasando por 1000 epocas con los datos de practica anteriormente ya mostrados.

```
model.compile(
    optimizer='adam',
    loss='mean_squared_error'
)

await ctx.send("La red neuronal esta procesando...")
historial = model.fit(xt, yt, epochs=1000, batch_size=4)
```

Presentamos la muestra de cómo es que va aprendiendo la red del bot en la siguiente imagen:



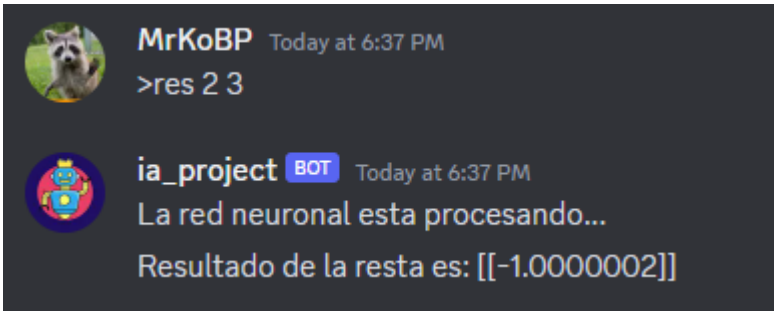
El Código del siguiente graficado es:

```
plt.xlabel("# epoca")
plt.ylabel("Magnitud de perdida")
plt.plot(historial.history['loss'])
plt.show()
```

Por último, realizamos la predicción y mostramos el resultado

```
xtest = np.array([[num1, num2]])

prediccion = model.predict(xtest)
await ctx.send("Resultado de la resta es: " + str(prediccion))
```

Ahora pasamos al chat de Discord y mostramos como se ve el resultado:



Realiza Multiplicación

Tenemos los datos de entrenamiento para la Red Neuronal

```
xt = np.array([[0.0, 0.0], [0.0, 1.0], [1.0, 0.0], [0.0, 20.0],
               [1.0, 1.0], [1.0, 3.0], [6.0, 1.0], [17.0, 1.0],
               [2.0, 2.0], [2.0, 4.0], [8.0, 2.0], [15.0, 2.0],
               [3.0, 3.0], [3.0, 5.0], [7.0, 3.0], [12.0, 3.0],
               [4.0, 4.0], [4.0, 2.0], [5.0, 4.0], [22.0, 4.0],
               [5.0, 5.0], [5.0, 6.0], [9.0, 5.0], [19.0, 5.0],
               [6.0, 6.0], [6.0, 7.0], [4.0, 6.0], [10.0, 6.0],
               [7.0, 7.0], [7.0, 2.0], [3.0, 7.0], [13.0, 7.0],
               [8.0, 8.0], [8.0, 9.0], [2.0, 8.0], [30.0, 8.0],
               [9.0, 9.0], [9.0, 8.0], [0.0, 9.0], [16.0, 9.0]])

yt = np.array([[0.0], [0.0], [0.0], [0.0],
               [1.0], [3.0], [6.0], [17.0],
               [4.0], [8.0], [16.0], [30.0],
               [9.0], [15.0], [21.0], [36.0],
               [14.0], [8.0], [20.0], [88.0],
               [25.0], [30.0], [45.0], [95.0],
               [36.0], [42.0], [24.0], [60.0],
               [49.0], [14.0], [21.0], [91.0],
               [64.0], [72.0], [16.0], [240.0],
               [81.0], [72.0], [0.0], [144.0]])
```

Tenemos 3 capas donde estas tienen el nombre de oculta1 esta contiene 100 neuronas y se le indica que el ingreso principal va a ser 2 datos, oculta2 está de la misma manera se le indica que tendrá 100 neuronas y por último tenemos la capa de salida que este contara con una única neurona, todo esto agregándolo al modelo con el que se realizara el aprendizaje.

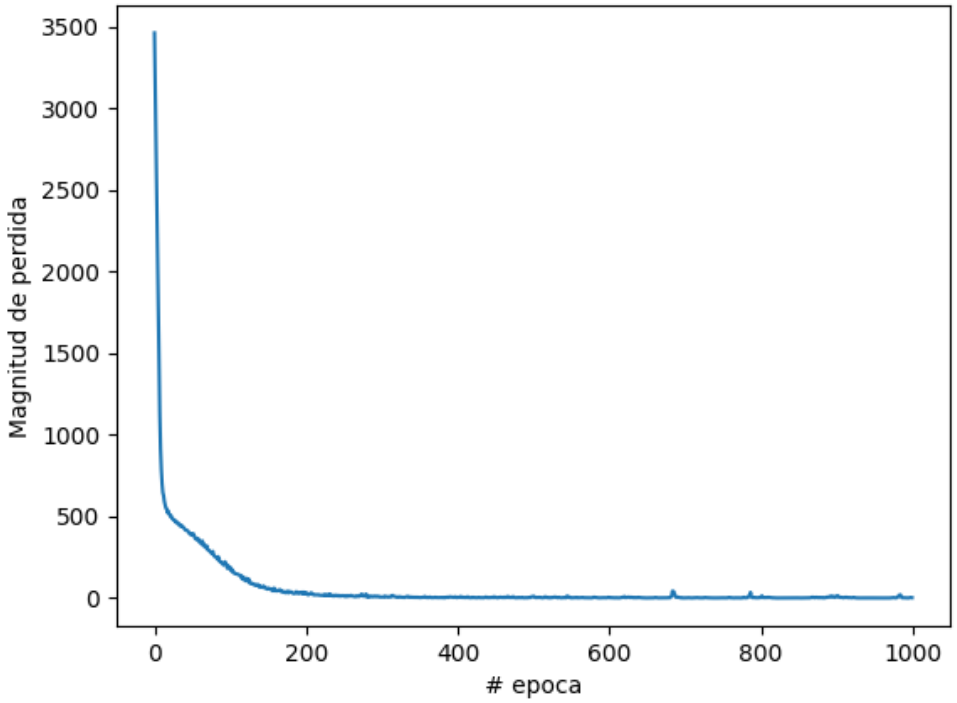
```
oculta1 = tf.keras.layers.Dense(units=100, input_shape=(2,), activation='relu')
oculta2 = tf.keras.layers.Dense(units=100, activation='relu')
salida = tf.keras.layers.Dense(units=1, activation='linear')
model = tf.keras.Sequential([oculta1, oculta2, salida])
```

Luego de tener el modelo listo lo compilamos con el optimizador Adam y le verificamos que el método de perdida sea mean_squared_error.

Seguido de enviamos el mensaje de que la red neuronal está trabajando y empezamos el método de enseñanza para la red neuronal pasando por 1000 épocas con los datos de practica anteriormente ya mostrados.

```
model.compile(
    optimizer='adam',
    loss='mean_squared_error'        You, 2 days ago • Agr
)

await ctx.send("La red neuronal esta procesando...")
historial = model.fit(xt, yt, epochs=1000, batch_size=4)
```

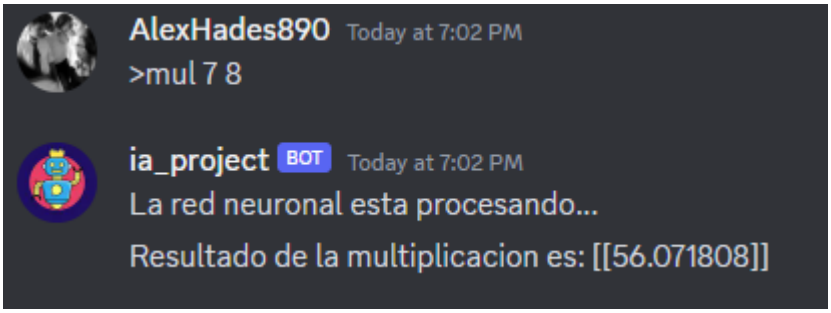
Presentamos la muestra de cómo es que va aprendiendo la red del bot en la siguiente imagen:



El Código del siguiente graficado es:

```
plt.xlabel("# epoca")
plt.ylabel("Magnitud de perdida")
plt.plot(historial.history['loss'])
plt.show()
```

Ahora pasamos al chat de Discord y mostramos como se ve el resultado:



Realiza División

Tenemos los datos de entrenamiento para la Red Neuronal

```
xt = np.array([[0.0, 0.0], [0.0, 1.0], [1.0, 0.0], [0.0, 20.0],
               [1.0, 1.0], [1.0, 3.0], [6.0, 1.0], [17.0, 1.0],
               [2.0, 2.0], [2.0, 4.0], [8.0, 2.0], [15.0, 2.0],
               [3.0, 3.0], [3.0, 5.0], [7.0, 3.0], [12.0, 3.0],
               [4.0, 4.0], [4.0, 2.0], [5.0, 4.0], [22.0, 4.0],
               [5.0, 5.0], [5.0, 6.0], [9.0, 5.0], [19.0, 5.0],
               [6.0, 6.0], [6.0, 7.0], [4.0, 6.0], [10.0, 6.0],
               [7.0, 7.0], [7.0, 2.0], [3.0, 7.0], [13.0, 7.0],
               [8.0, 8.0], [8.0, 9.0], [2.0, 8.0], [30.0, 8.0],
               [9.0, 9.0], [9.0, 8.0], [0.0, 9.0], [16.0, 9.0]])

yt = np.array([[0.0], [0.0], [0.0], [0.0],
               [1.0], [0.3], [6.0], [17.0],
               [1.0], [0.5], [4.0], [7.5],
               [1.0], [0.6], [2.3], [4.0],
               [1.0], [2.0], [1.25], [5.5],
               [1.0], [0.5], [1.8], [3.8],
               [1.0], [0.85], [0.67], [1.67],
               [1.0], [3.5], [0.43], [1.85],
               [1.0], [0.89], [0.25], [3.75],
               [1.0], [1.125], [0.0], [1.77]])
```

Tenemos 3 capas donde estas tienen el nombre de oculta1 esta contiene 100 neuronas y se le indica que el ingreso principal va a ser 2 datos, oculta2 está de la misma manera se le indica que tendrá 100 neuronas y por último tenemos la capa de salida que este contara con una única neurona, todo esto agregándolo al modelo con el que se realizara el aprendizaje.

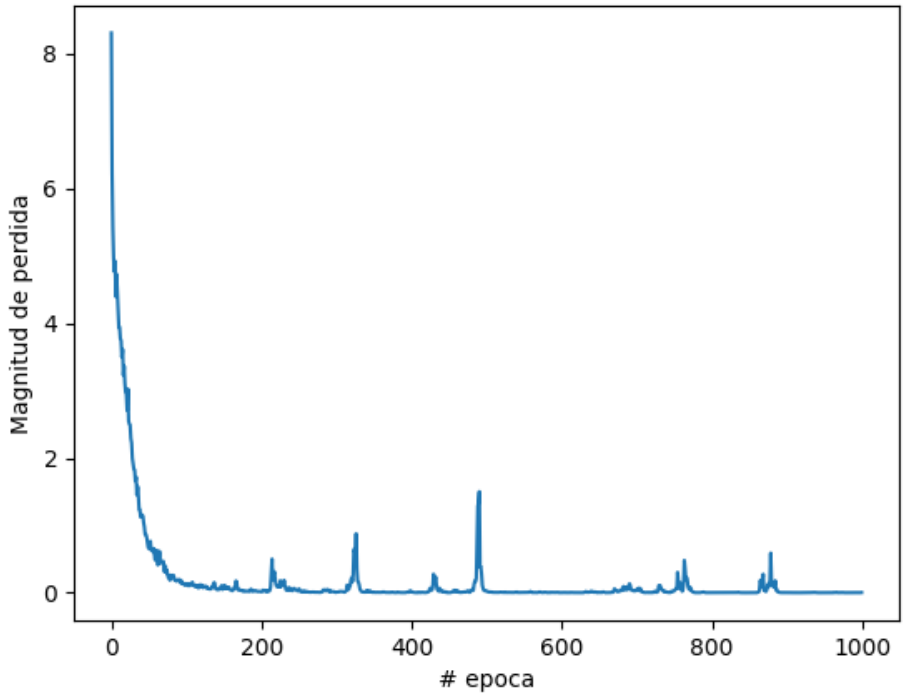
```
oculta1 = tf.keras.layers.Dense(units=100, input_shape=(2,), activation='relu')
oculta2 = tf.keras.layers.Dense(units=100, activation='relu')
salida = tf.keras.layers.Dense(units=1, activation='linear')
model = tf.keras.Sequential([oculta1, oculta2, salida])
```

Luego de tener el modelo listo lo compilamos con el optimizador Adam y le verificamos que el método de perdida sea mean_squared_error.

Seguido de enviamos el mensaje de que la red neuronal está trabajando y empezamos el método de enseñanza para la red neuronal pasando por 1000 épocas con los datos de practica anteriormente ya mostrados.

```
model.compile(
    optimizer='adam',
    loss='mean_squared_error'|      You, 2 days ago • Agr
)

await ctx.send("La red neuronal esta procesando...")
historial = model.fit(xt, yt, epochs=1000, batch_size=4)
```

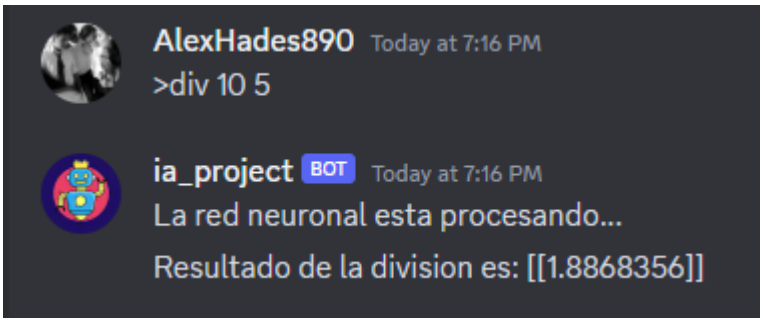
Presentamos la muestra de cómo es que va aprendiendo la red del bot en la siguiente imagen:



El Código del siguiente graficado es:

```
plt.xlabel("# epoca")
plt.ylabel("Magnitud de perdida")
plt.plot(historial.history['loss'])
plt.show()
```

Ahora pasamos al chat de Discord y mostramos como se ve el resultado:



Convierte grados de Celsius a Fahrenheit

Tenemos los datos de entrenamiento para la Red Neuronal

```
celsius = np.array([-40, -10, 0, 8, 15, 22, 38], dtype=float)
fahrenheit = np.array([-40, 14, 32, 46, 59, 72, 100], dtype=float)
```

Tenemos 3 capas donde estas tienen el nombre de oculta1 esta contiene 3 neuronas y se le indica que el ingreso principal va a ser 1 dato, oculta2 está de la misma manera se le indica que tendrá 3 neuronas y por último tenemos la capa de salida que este contara con una única neurona, todo esto agregándolo al modelo con el que se realizara el aprendizaje.

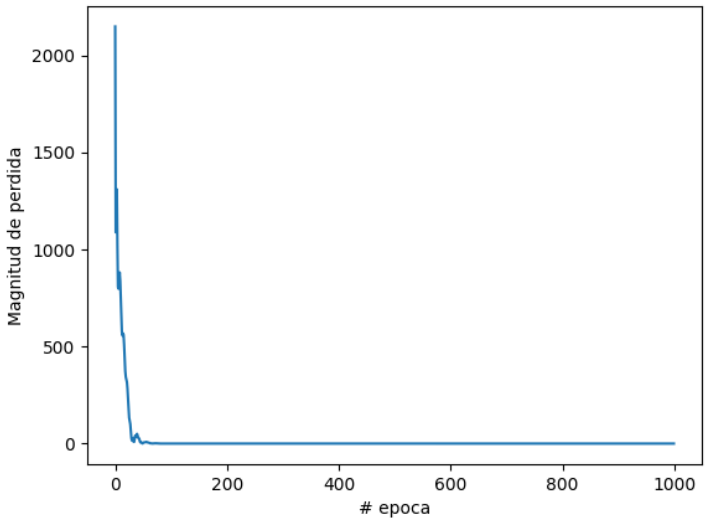
```
oculta1 = tf.keras.layers.Dense(units=3, input_shape=[1])
oculta2 = tf.keras.layers.Dense(units=3)
salida = tf.keras.layers.Dense(units=1)
model = tf.keras.Sequential([oculta1, oculta2, salida])
```

Luego de tener el modelo listo lo compilamos con el optimizador Adam y le verificamos que el método de perdida sea mean_squared_error.

Seguido de enviamos el mensaje de que la red neuronal está trabajando y empezamos el método de enseñanza para la red neuronal pasando por 1000 épocas con los datos de practica anteriormente ya mostrados.

```
model.compile(
    optimizer=tf.keras.optimizers.Adam(0.1),
    loss='mean_squared_error'
)

await ctx.send("La red neuronal esta procesando...")
historial = model.fit(celsius, fahrenheit, epochs=1000, verbose=False)
```

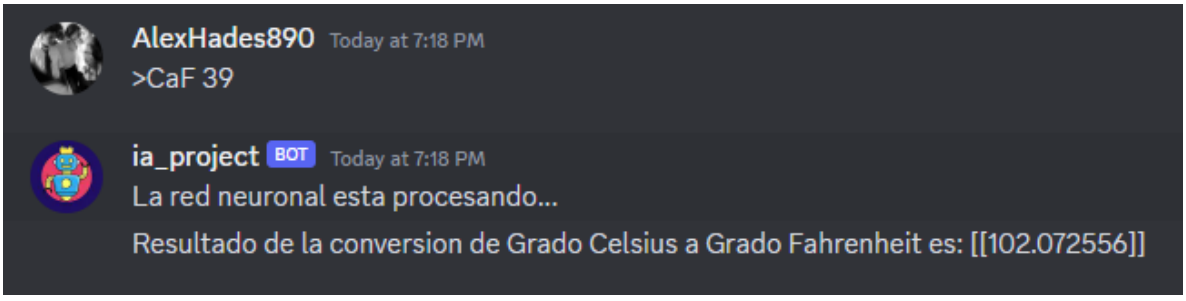
Presentamos la muestra de cómo es que va aprendiendo la red del bot en la siguiente imagen:



El Código del siguiente graficado es:

```
plt.xlabel("# epoca")
plt.ylabel("Magnitud de perdida")
plt.plot(historial.history['loss'])
plt.show()
```

Ahora pasamos al chat de Discord y mostramos como se ve el resultado:



Convierte grados de Fahrenheit a Celsius

Tenemos los datos de entrenamiento para la Red Neuronal

```
celsius = np.array([-40, -10, 0, 8, 15, 22, 38], dtype=float)
fahrenheit = np.array([-40, 14, 32, 46, 59, 72, 100], dtype=float)
```

Tenemos 3 capas donde estas tienen el nombre de oculta1 esta contiene 3 neuronas y se le indica que el ingreso principal va a ser 1 dato, oculta2 está de la misma manera se le indica que tendrá 3 neuronas y por último tenemos la capa de salida que este contara con una única neurona, todo esto agregándolo al modelo con el que se realizará el aprendizaje.

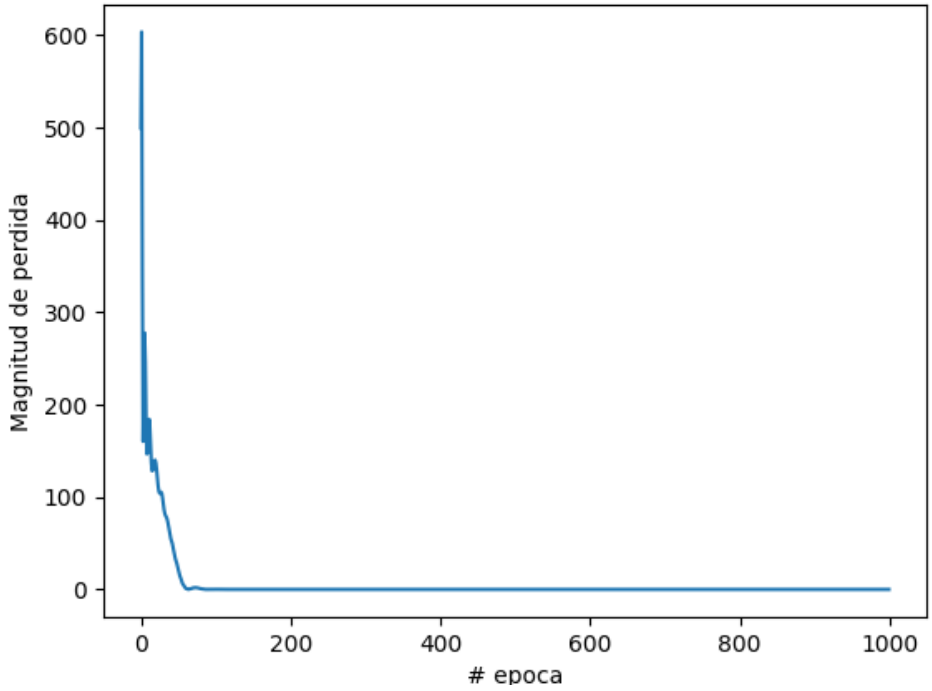
```
oculta1 = tf.keras.layers.Dense(units=3, input_shape=[1])
oculta2 = tf.keras.layers.Dense(units=3)
salida = tf.keras.layers.Dense(units=1)
model = tf.keras.Sequential([oculta1, oculta2, salida])
```

Luego de tener el modelo listo lo compilamos con el optimizador Adam y le verificamos que el método de perdida sea mean_squared_error.

Seguido de enviamos el mensaje de que la red neuronal está trabajando y empezamos el método de enseñanza para la red neuronal pasando por 1000 épocas con los datos de practica anteriormente ya mostrados.

```
model.compile(
    optimizer=tf.keras.optimizers.Adam(0.1),
    loss='mean_squared_error'
)

await ctx.send("La red neuronal esta procesando...")
historial = model.fit(fahrenheit, celsius, epochs=1000, verbose=False)
```

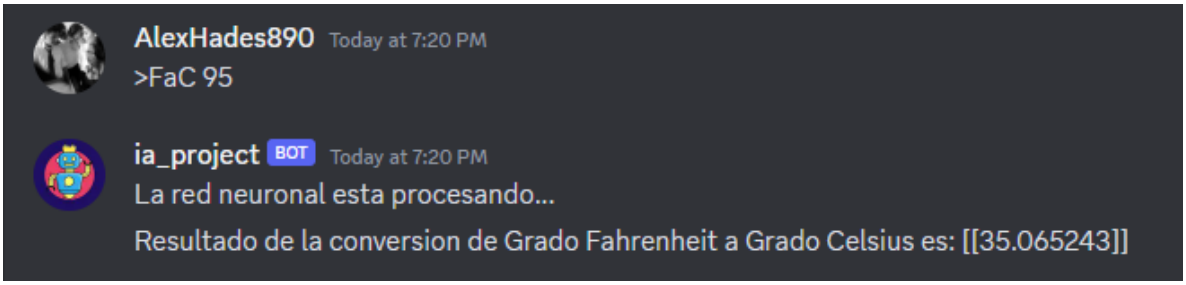
Presentamos la muestra de cómo es que va aprendiendo la red del bot en la siguiente imagen:



El Código del siguiente graficado es:

```
plt.xlabel("# epoca")
plt.ylabel("Magnitud de perdida")
plt.plot(historial.history['loss'])
plt.show()
```

Ahora pasamos al chat de Discord y mostramos como se ve el resultado:



Reconocimiento de perro o gato

Este será un proyecto en donde se realiza una red neuronal capaz de identificar ya sea perros o gatos en donde la codificación de este la realizamos de la siguiente manera:

Iniciamos importando las librerías que vamos a utilizar como lo son tensorflow que es el que utilizaremos para crear la red neuronal, tensorflow_datasets es para obtener la información con el que realizaremos el entrenamiento a la red neuronal, matplotlib es la librería que utilizaremos para la graficacion del modelo de aprendizaje como va transcurriendo, cv2 para modificar el tamaño y el color de las imágenes y por ultimo numpy para utilizar los arreglos de esta librería para una mejor velocidad de procesamiento.

Como vemos en la siguiente foto estamos obteniendo la información de cats_vs_dogs en la variable metadatos.

```
import tensorflow as tf
import tensorflow_datasets as tfds
import matplotlib.pyplot as plt
import cv2
import numpy as np

datos, metadatos = tfds.load('cats_vs_dogs', as_supervised=True, with_info=True)

TAMANO_IMG = 100

datos_entrenamiento = []
x = [] #entrada
y = [] #etiqueta (perro o gato)
```

En la siguiente foto obtenemos los datos de entrenamiento mediante un ciclo for que contiene, imagen y etiqueta, en lo que en la variable imagen rehacemos el tamaño de la imagen con cv2 haciéndola de 100 x 100, seguido de colocamos con la ayuda de cv2 la imagen en escala de grises esto para facilitarle a la red neuronal la obtención de información, a esto se le llama adelgazamiento de imagen, seguido de agregamos los datos en el arreglo de

datos_entrenamiento, y por ultimo volvemos a rehacer el tamaño de la imagen en 100 x 100 pero agregamos que es de un solo canal el color ósea solo escala de grises, de la misma manera separamos la imagen y etiquetas y guardamos en las variables x, y.

```
for i, (imagen, etiqueta) in enumerate(datos['train']):
    imagen = cv2.resize (imagen.numpy(), (TAMANO_IMG, TAMANO_IMG))
    imagen = cv2.cvtColor(imagen, cv2.COLOR_BGR2GRAY)
    datos_entrenamiento.append([imagen, etiqueta])
    imagen = imagen.reshape(TAMANO_IMG, TAMANO_IMG, 1)
    x.append(imagen)
    y.append(etiqueta)

x = np.array(x).astype(float) / 255
y = np.array(y)
```

Seguido de agregamos una función que trae la librería tensorflow que ayuda a hacer que la imagen obtenga cambios como rotado, movido, tapado, con zoom, para mostrarle a la red artificial que también pueden llegarle imágenes de este estilo.

```
datagen = tf.keras.preprocessing.image.ImageDataGenerator(
    rotation_range=30,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=15,
    zoom_range=[0.7, 1.4],
    horizontal_flip=True,
    vertical_flip=True
)

datagen.fit(x)
```

Seguido de mostramos las imágenes ya modificadas como dicho en el punto anterior.

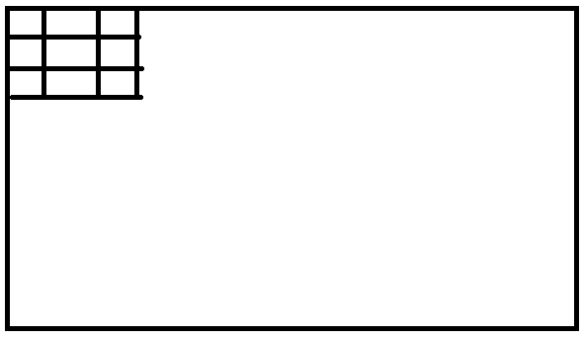
Codigo

```
plt.figure(figsize=(20,8))

for imagen, etiqueta in datagen.flow(x, y, batch_size=10, shuffle=False):
    for i in range(10):
        plt.subplot(2, 5, i+1)
        plt.xticks([])
        plt.yticks([])
        plt.imshow(imagen[i].reshape(100, 100), cmap="gray")
    break
```

Imagen obtenida

Seguido del punto anterior realizamos un modelo de una red neuronal convolucionales,

En donde agregamos la primera capa la agregamos de 32 neuronas que van a ingresar imágenes de 100 x 100 en escala de grises y va a buscar en recuadros de 3 x 3 ¿qué significa esto? Que va a tomar pixel por pixel de la imagen y los va a tomar de 3 x 3 ejemplo



Va tomar cada pixel y va a realizar un grupo de pixeles así para tomar el pixel mas grande esto para poder hacer más pequeña la imagen y la red neuronal pueda trabajar de mejor manera con ella.

Seguido tenemos otra capa igual a la anterior con la diferencia que acá ya no hay ingreso y esta contiene 64 neuronas.

Seguido tenemos otra capa igual a la anterior con la diferencia que esta ya no busca en el recuadro de 3 x 3 y esta contiene 100 neuronas.

Por ultima tenemos la capa de salida, en donde únicamente tiene 1 sola neurona, y tenemos que es de tipo sigmoid, ¿que significa que sea sigmoid? Que esta va a retornar valores que estén entre 0 y 1.

```
modeloCNN_AD = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(100, 100, 1)),
    tf.keras.layers.MaxPooling2D(2, 2),

    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),

    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(100, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

Agregamos el optimizador Adam, método de aprendizaje de perdida de datos y las métricas, para compilar.

```
modeloCNN_AD.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy']
)
```

Separamos las variables de entrenamiento y las de test para poder probarlo después.

```
x_entrenamiento = x[:19700]
x_test = x[19700:]

y_entrenamiento = y[:19700]
y_test = y[19700:]
```

Y por último corremos el modelo para que aprenda los datos con 100 épocas, este método es muy tardado.

```
data_gen_entrenamiento = datagen.flow(x_entrenamiento, y_entrenamiento, batch_size=32)

tensorboardDenso_AD = tf.keras.callbacks.TensorBoard(log_dir='logs/denso_AD')

modeloCNN_AD.fit(
    data_gen_entrenamiento,
    epochs=100, batch_size=32,
    validation_data=(x_test, y_test),
    steps_per_epoch=int(np.ceil(len(x_entrenamiento) / float (32))),
    validation_batch_size=int(np.ceil(len(x_test) / float (32))),
    callbacks=[tensorboardDenso_AD]
)
```

Seguido de importamos el modelo y pasamos a abrir nuestro visual studio code y mostramos lo siguiente:

Hacemos un archivo index.html y iniciamos la creación normal de un codigo html, agregamos 2 canvas para programarlos con js para la utilización de la cámara, uno para la imagen que va a capturar y el otro para que lo compare con el modelo, que creamos con anterioridad.

```
<canvas id="canvas" width="400" height="400" style="max-width: 100%;"></canvas>
<canvas id="otrocanvas" width="150" height="150" style="display: none"></canvas>
```

Cargamos el modelo de la siguiente manera:

```
(async() => {
  console.log("Cargando modelo...");
  modelo = await tf.loadLayersModel("model.json");
  console.log("Modelo cargado");
})();
```

Y luego realizamos una función para predecir en donde tomamos la imagen del canvas y la pasamos al canvas más pequeño tomando 100 x 100 igual a la cual estábamos trabajando el modelo en Python.

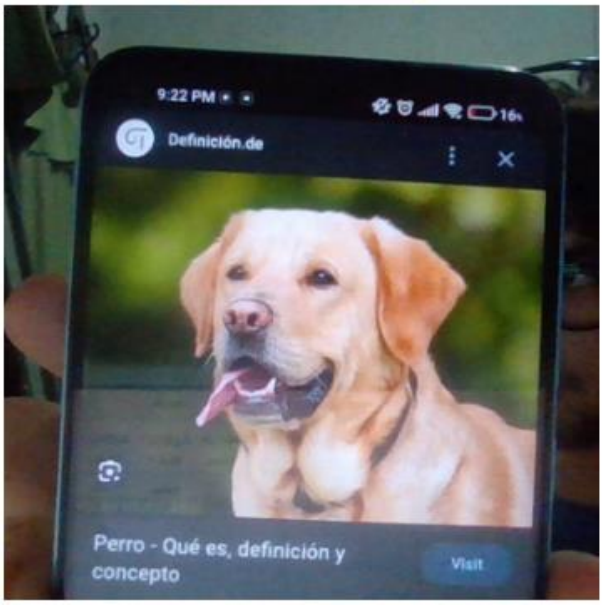
También le colocamos en escala de grises y lo mandamos al modelo de manera async para que espere el resultado, obtendrá una respuesta en donde si es menor o igual de 0.5 (lo que se había explicado anteriormente de valores entre 0 y 1) es un gato y lo contrario es un perro, este seguido de se mostrara en pantalla.

Este proceso se hará cada 150 milisegundos.

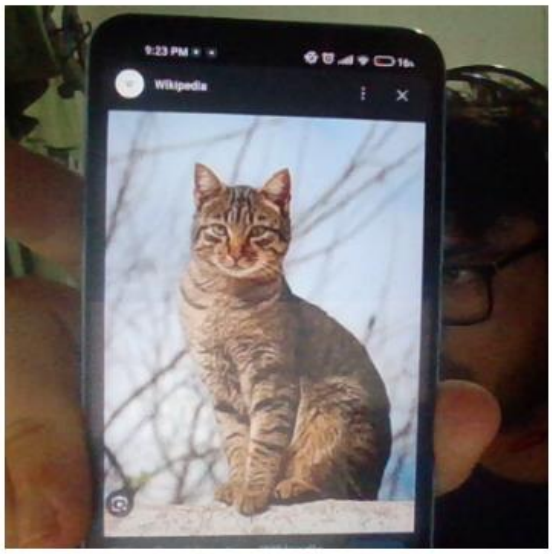
```
function predecir() {
  if (modelo != null) {
    resample_single(canvas, 100, 100, otrocanvas);

    //Hacer la predicción
    var ctx2 = otrocanvas.getContext("2d");
    var imgData = ctx2.getImageData(0,0, 100, 100);

    var arr = [];
    var arr100 = [];

    for (var p=0; p < imgData.data.length; p+= 4) {
      var rojo = imgData.data[p] / 255;
      var verde = imgData.data[p+1] / 255;
      var azul = imgData.data[p+2] / 255;

      var gris = (rojo+verde+azul)/3;

      arr100.push([gris]);
      if (arr100.length == 100) {
        arr.push(arr100);
        arr100 = [];
      }
    }

    arr = [arr];
```

```
    var tensor = tf.tensor4d(arr);
    var resultado = modelo.predict(tensor).dataSync();

    var respuesta;
    if (resultado <= .5) {
      respuesta = "Gato";
    } else {
      respuesta = "Perro";
    }
    document.getElementById("resultado").innerHTML = respuesta;

  }

  setTimeout(predecir, 150);
}
```

Por último, mostramos el resultado de este:



Perro



Gato

Repositorio: https://github.com/BPoER00/Bot-Discord-ia_project