Instituto Tecnológico de Costa Rica

Escuela de Ingeniería en Computación

Aplicación de deep learning al aprendizaje de modelos de mundo en mecanismos cognitivos

Segundo informe de práctica profesional

Bachillerato en Ingeniería en Computación

Ariel Antonio Rodríguez Jiménez

Coruña, España; mayo 2018

Tabla de contenido

Introducción

En el presente documento se mostrarán diagramas de clases y componentes pertenecientes a los programas realizados en la práctica profesional, tanto con TensorFlow como con la librería Python-Neat que hace uso de algoritmos evolutivos para encontrar una óptima arquitectura para una red neuronal. Además de los diagramas, hay información del proyecto como los riesgos del sistema y el plan de trabajo.

Es importante siempre tener una bitácora de los datos más relevantes de cada programa, por ello se crearon clases para ayudar al manejo de estos. Más adelante se mencionará cuáles son estos datos y la manera en la cual son almacenados para su visualización y análisis.

Modelo de diseño

Arquitectura conceptual de la solución

El problema que se está tratando es la predicción de una posición futura, tomando en cuenta una posición inicial y una actuación. Tanto entradas como salidas están hechas en coordenadas polares (distancia y ángulo). Hay un brazo robótico en una mesa y una bola (ambos ubicados en cualquier parte de la mesa), el objetivo es que el brazo coja la bola. Los inputs de la red son la distancia y ángulo del brazo a la bola, distancia y ángulo de actuación del brazo y se tiene que predecir la distancia y ángulo a la que queda el brazo cuando realiza la actuación.

Hasta ahora, se está teniendo problemas con las coordenadas polares pues el error es muy alto, por ello se hizo un cambio de los datos a coordenadas cartesianas y el resultado ha sido satisfactorio, el error mejoró de una manera significativa. Sin embargo, tanto con el algoritmo Neat como con Tensorflow, se está intentando resolver el problema con coordenadas polares.

Tensorflow

La arquitectura de la red neuronal que se está utilizando para la predicción (con coordenadas cartesianas) es una capa intermedia con 15 neuronas y sin función de activación, 4 entradas y 2 salidas, 40 registros de entrenamiento (este número varía) y 500000 de test final. El error cuadrático medio del test es de ~0.00000001. No se utiliza función de activación pues lo que tiene que hacer la red es una resta, o sea no hay alguna no linealidad.

Por otra parte, con las coordenadas polares aún se están probando distintas arquitecturas con funciones de activación distintas (sigmoid, relu, tanh; dependiendo de la normalización de los datos de entrada).

Python-Neat

Neat es un algoritmo evolutivo que busca una arquitectura de red óptima que resuelva el problema. Hasta el momento el mejor error que ha dado es de ~0.003

(test final) utilizando como entradas y salidas las coordenadas cartesianas, con 7000 generaciones y una población de 300 individuos. Y ~0.09 utilizando las coordenadas polares. La idea es obtener la arquitectura óptima que encuentre el algoritmo y replicarla en Tensorflow para hacer un entrenamiento de esa red.

Bitácora

Al final de cada entrenamiento los errores de training y testing son almacenados en un csv para poder graficarlos y ver el comportamiento de la red. Además, con Tensorboard se puede ver la evolución de los pesos de las capas intermedias durante su entrenamiento. Al final de cada entrenamiento siempre se realiza un test, y al final todas las predicciones con sus respectivas respuestas correctas, son almacenadas en un archivo csv.

Interfaz de usuario

La interfaz de usuario es la consola o terminal donde quiera ejecutar los programas, esto debido a que no es necesaria una GUI para el buen funcionamiento del programa. En el caso de Tensorflow, en la terminal se imprime la iteración por la que va el programa, el error del training y del testing respectivamente (Imagen 1). Con Python-Neat, en la terminal se imprime la generación, la media, desviación y el mejor de los fitness de los individuos (Imagen 2).

Por otra parte, el programa de red neuronal genera 3 archivos csv que contienen una bitácora de los errores por iteración, pesos de las capas ocultas (el usuario escoge los momentos en los que se guardan y es opcional) y otro con predicciones realizadas por la red neuronal; cada línea del archivo de predicciones contiene la predicción hecha por la red y su respectiva respuesta correcta. Ejemplo: predicción_1, respuesta_correcta_1.

La visualización de los pesos se puede realizar con Tensorboard en el navegador de la computadora, sin embargo, el almacenamiento de los datos hace que el programa aumente su tiempo de ejecución de una manera significativa. Una vez

termina el entrenamiento, se lee el csv de los errores y se muestra una gráfica con estos. (Imagen 6, Imagen 7)

Diagrama de clases

Los diagramas de clases se encuentran adjuntos en la sección de anexos para una mejor visualización (Imagen 3, Imagen 4, Imagen 5).

Explicación de las clases

PersistanceManager

Guarda y restaura las variables que el usuario quiera en un programa de Tensorflow, por ejemplo: los pesos y biases de cada capa de una red neuronal. Esto lo hace mediante la clase tf.train.Saver() que provee Tensorflow.

Dataset

Esta clase es abstracta. Carga un archivo y almacena su contenido en un atributo. La idea es que la clase sea tratada como un iterador una vez sean cargados los registros. Como ahora estamos realizando dos tipos de entrenamiento: clásico e incremental (Incrementa N registros al dataset cada iteración. Con esto simula la manera en la cual le llegarían los datos en tiempo real); se crearon dos clases abstractas más para diferenciar el comportamiento de ambos dataset. También es utilizado para cargar los datasets de entrenamiento y testing en el programa con el algoritmo Neat.

ClassicDataset

Clase abstracta, hereda de Dataset. Esta clase itera sobre los registros de una manera común, el dataset se puede fragmentar en batches y se puede hacer shuffle de los datos. Mediante el método *dataset_out_of_range ()* se verifica que el iterador no haya recorrido todo el dataset y pueda hacerse uso del método *get_next()* para solicitar otro registro/batch. Cuando se itera sobre todos los registros, se puede utilizar el método *restore_index ()* para comenzar de nuevo.

IncrementalDataset

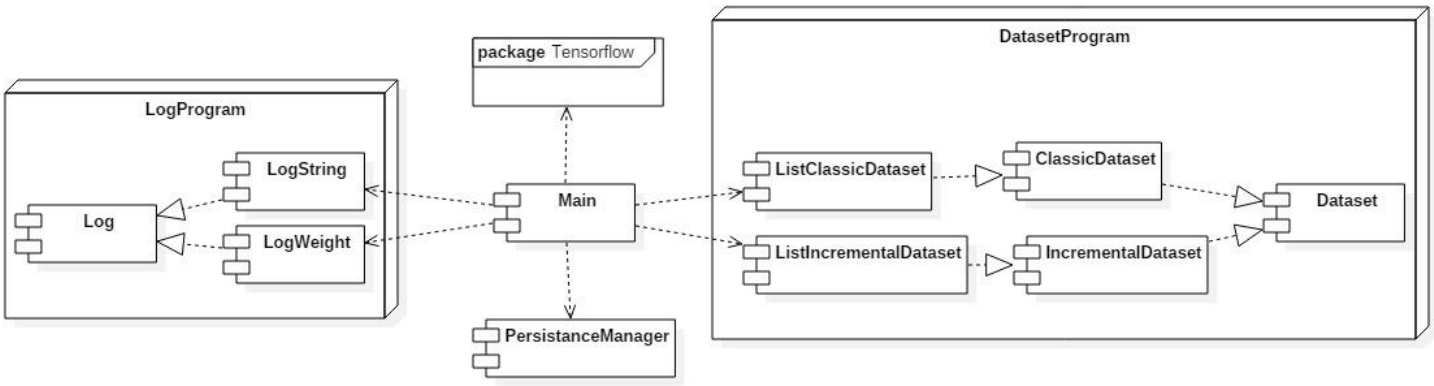
Clase abstracta, hereda de Dataset. Esta clase itera sobre los registros de una misma manera que el ClassicDataset, incluso sus métodos de acceso a los registros son los mismos. La diferencia es que le agrega los métodos *increment_dataset ()* e *increment_out_of_range ()* en los cuales incrementa el dataset en el número de registros que se define al crear la clase y verifica que aún queden registros por incrementar, respectivamente. El dataset siempre comienza con el número de registros definido por el usuario en el constructor (*increment_size*). Además, en esta clase no se pueden hacer batches.

Log

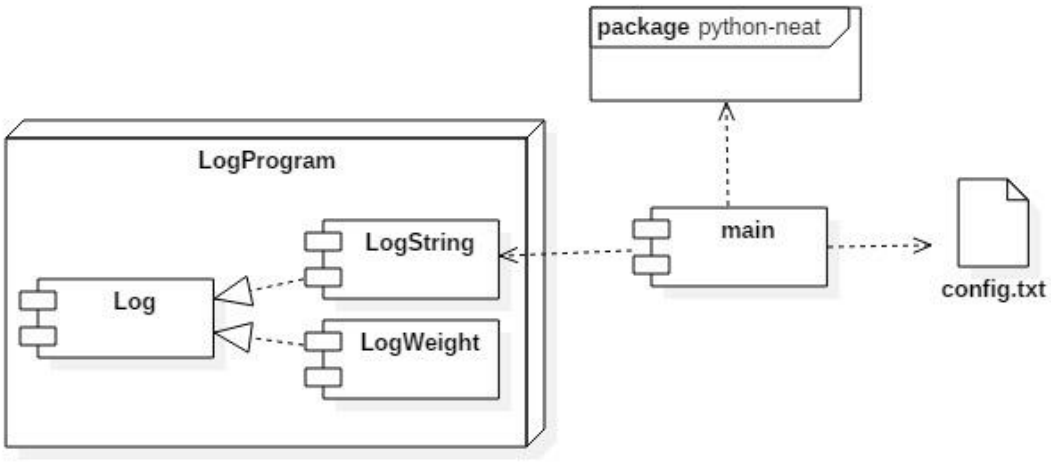
Es utilizada para almacenar datos en un archivo en disco. Es una clase abstracta porque el método de almacenar datos o log puede tener distintos tipos de parámetros dependiendo de los que se quiera almacenar y la manera en la que se quiera guardar. Por otra parte, los métodos *save ()*, guarda los datos en disco, y *close_file ()*, cierra el archivo de bitácora; estos sí son implementados por esta clase porque se ejecutan siempre de una misma manera.

Diagrama de componentes

Programa con Tensorflow



Programa con Python-Neat



Plan de trabajo

El plan de trabajo que queda para las próximas semanas es el siguiente, según el original.

Actividad	Semanas
Integrar la técnica más satisfactoria en el MDB**.**	2
Analizar el MDB mejorado en un problema de aprendizaje en tiempo real**.**	4

Análisis de riesgos

1) Implementación sin funcionar
    a) Categoría
        i) Tecnológico
    b) Posibles causas
        i) Deficiente programación de parte del estudiante
        ii) Deficiente propuesta de solución
    c) Impacto del riesgo para el proyecto
        i) Tiempo perdido intentando solucionar problemas con nueva tecnología
    d) Probabilidad de ocurrencia
        i) 0.1
    e) Estrategia de evasión
        i) Revisar constantemente los resultados obtenidos
    f) Estrategia de mitigación
        i) Corregir errores encontrados cada vez que se revisen los resultados
    g) Estrategia de contingencia
        i) Seguir con los programas anteriores y hacer análisis de los errores obtenidos para encontrar una solución al respecto.

Anexos

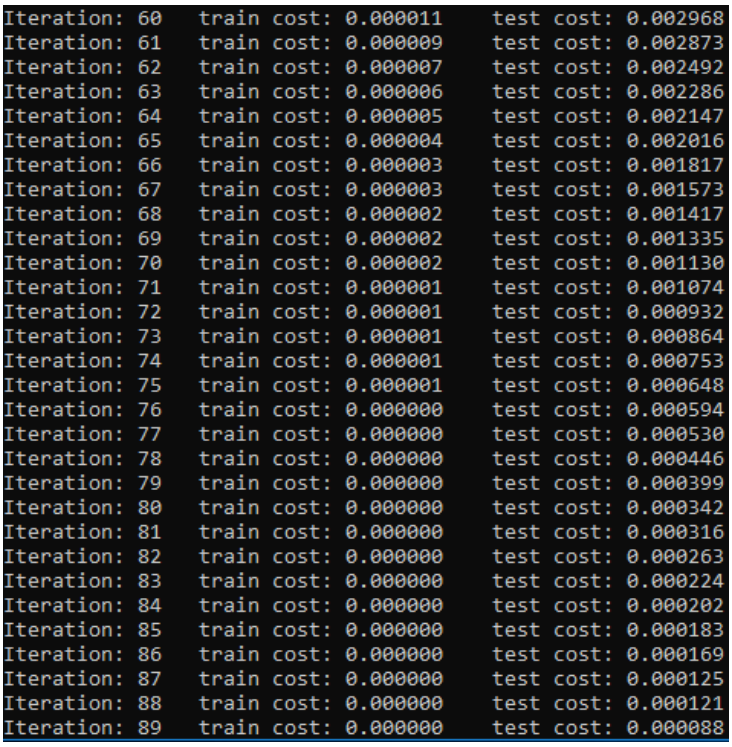


Imagen 1: Ejemplo de interfaz de usuario al ejecutar el entrenamiento de la red neuronal.

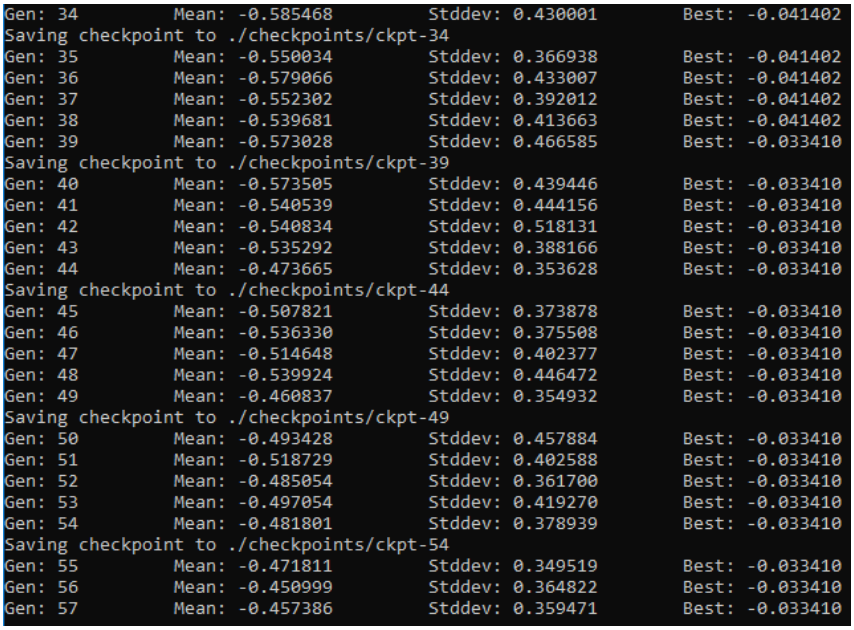


Imagen 2: Ejemplo de la interfaz de usuario al ejecutar el algoritmo evolutivo NEAT.

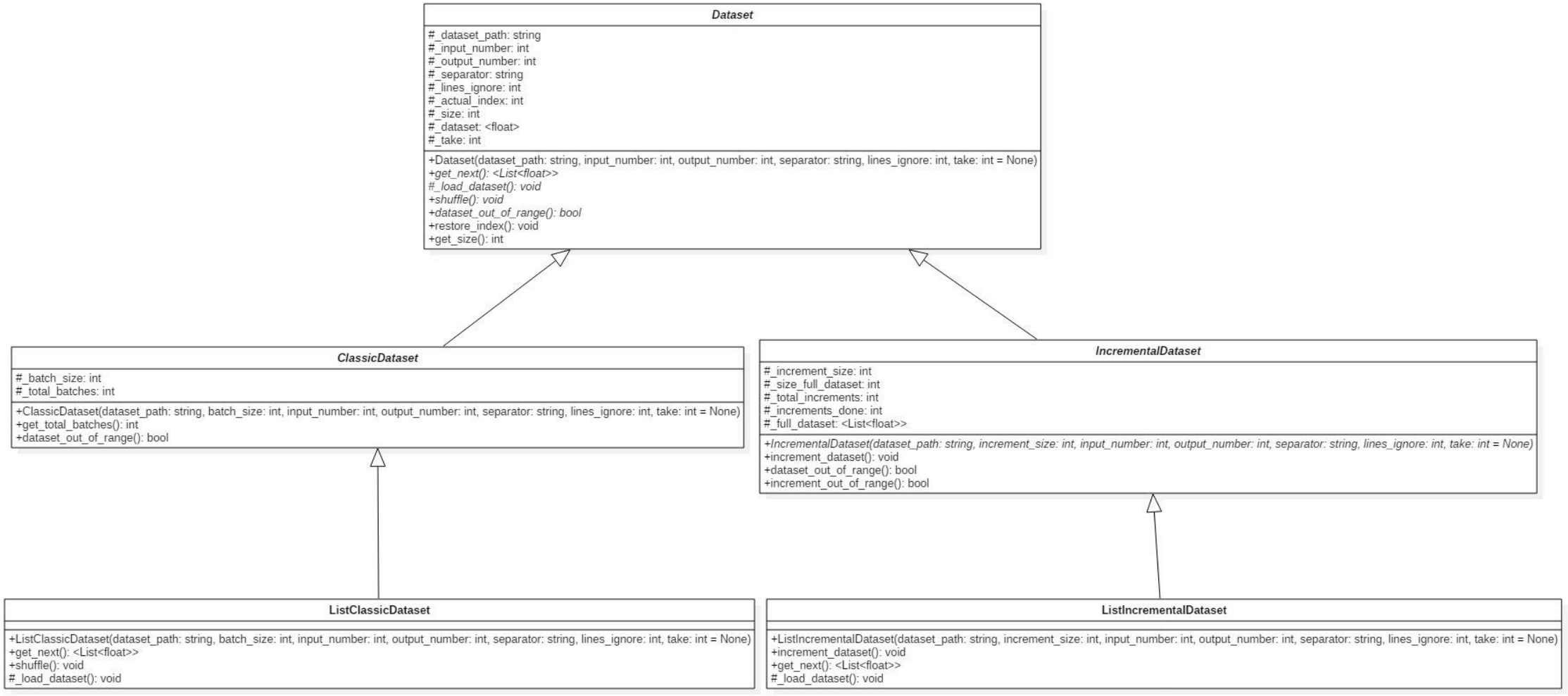


Imagen 3: Clase utilizada para cargar e iterar sobre los dataset.

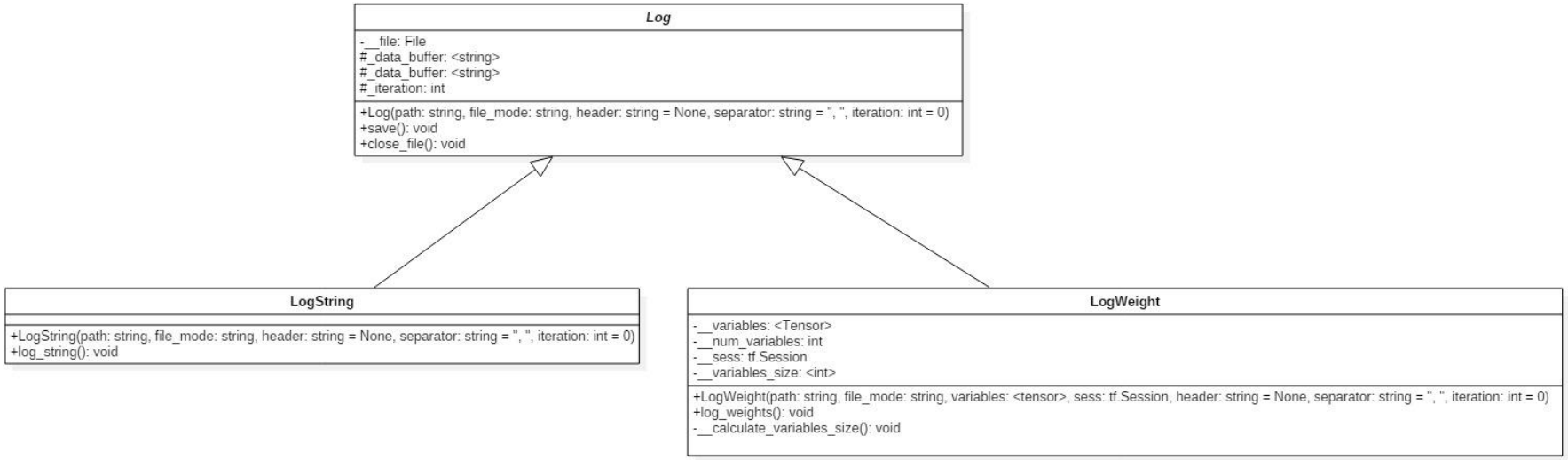


Imagen 4: Clase utilizada para guardar datos como bitácora.

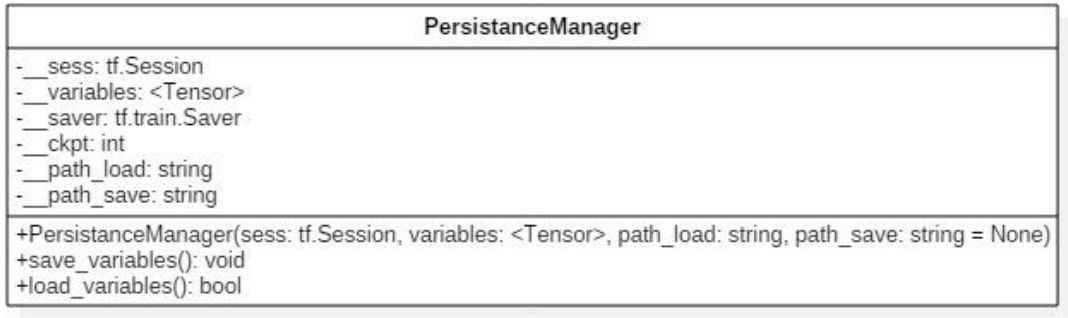


Imagen 5: Clase utilizada para la persistencia de datos de las variables de Tensorflow.

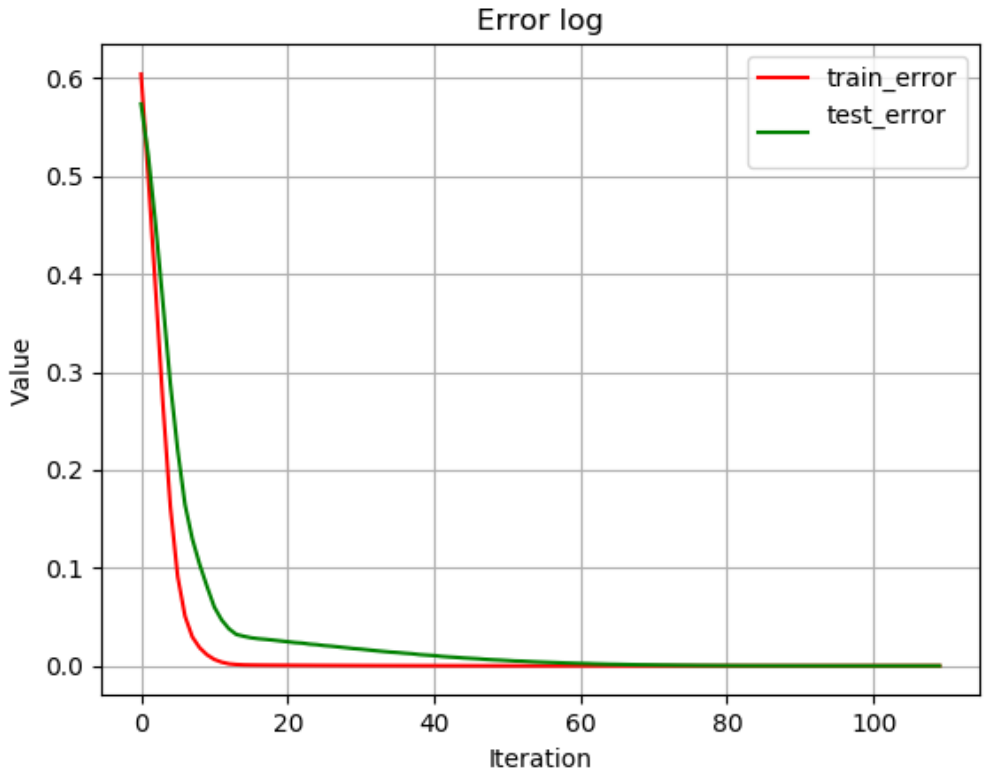


Imagen 6: Gráfico de los errores después de un entrenamiento de una red neuronal con Tensorflow.

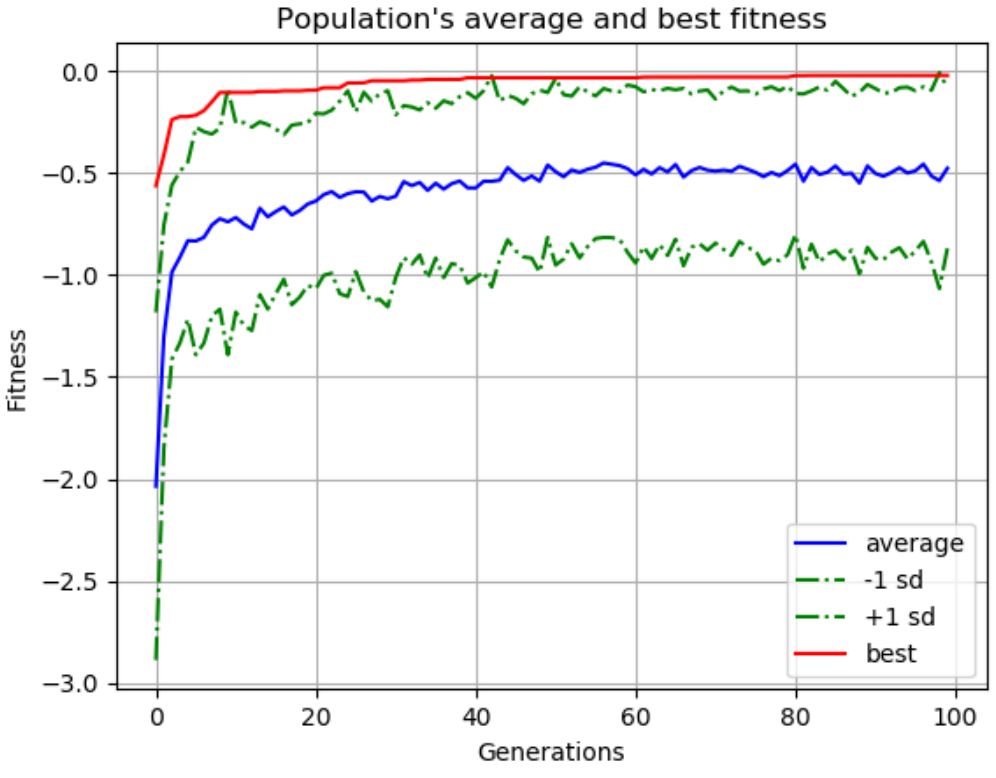


Imagen 7: Gráfico con los datos estadísticos de los fitness de los individuos por generación.