Taller 2 Redes Neuronales - Aprendizaje de Máquina

Daniel Andrés Crovo Pérez, *dcrovo@javeriana.edu.co*, Santiago Salazar, *agustin.salazar@javeriana.edu.co*

***Abstract*—En este documento se ilustra el proceso de desarrollo del Taller 2 de la asignatura Aprendizaje de Máquina, en el cual se trabajaron los conceptos básicos de las redes neuronales feed-forward, su implementación utilizando Tensorflow y aplicación para problemas de clasificación.**

***Index Terms*—Machine Learning, Artificial Neural Networks.**

I. Introducción

EL Taller consiste de 2 puntos desarrollables, y previamente se realiza una guía paso a paso para la generación de datasets y entrenamiento de redes neuronales con Tensorflow. El presente informe se dividirá en dos partes que corresponden a el desarrollo de cada punto.

II. Punto 1

. Esta red aún no presenta un buen comportamiento en el conjunto de datos. Explore los hiperparámetros de la red para realizar una clasificación adecuada, mida el performance utilizando métricas de clasificación. Justifique los pasos que lo llevaron a escoger el modelo utilizando las curvas del proceso de aprendizaje. Grafique las fronteras de decisión encontradas.

A. Dataset

El conjunto de datos se generó por medio de una función la cual genera dos espirales en sentido contrario que corresponden a 2 clases diferentes.

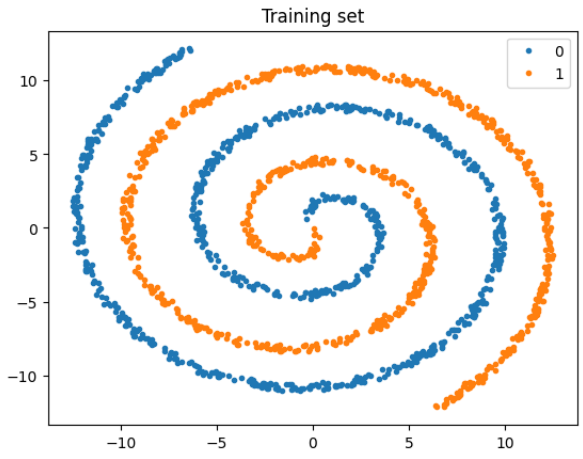

Fig. 1. Conjunto de datos para clasificación.

Como se puede observar el patrón de cada clase es no lineal, por lo cual una red neuronal puede suponer una forma apropiada para realizar la clasificación de las instancias. El conjunto de datos se separa en entrenamiento y validación, obteniendo 1600 y 400 instancias respectivamente.

B. Modelo desarrollado

Inicialmente, se implementa un modelo con una capa oculta y 8 unidades neuronales, el cual al ser muy simple no realiza un buen trabajo de clasificación, obteniendo una exactitud de 0.55 en el conjunto de validación. Posteriormente, se procedió a implementar un modelo de dos capas ocultas cada una con 4 unidades neuronales, el cual fue entrenado durante 300 épocas, sin embargo, obtuvo el mismo resultado que el anterior.

Finalmente, se planteó un experimento inicial con un modelo de 3 capas ocultas y 32, 16, 8 unidades neuronales. Todas las capas tienen una función de activación relu, y la capa de salida sigmoidea. El modelo se entrenó con el optimizador Adam, una taza de aprendizaje de 0.01, y la función de pérdida utilizada fue *binary cross entropy*. Adicionalmente, se hizo uso de callbacks para para el entrenamiento de forma automática cuando se detecte que la función de pérdida en el conjunto de validación no mejora por más de 10 épocas.

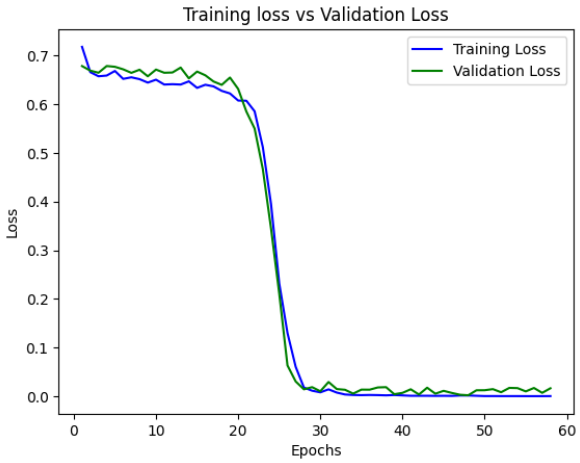

Fig. 2. Pérdida en el conjunto deentrenamiento y validación.

Como se puede observar en la

III. Punto 2

. El objetivo principal era desarrollar y optimizar un modelo de redes neuronales para predecir la variable objetivo *label*. A continuación, se describe el proceso detallado:

1) **Preprocesamiento de datos**:
   - *Carga de Datos*: Se importaron los datos del archivo *public_dataset.csv* utilizando la biblioteca `pandas`. Este conjunto de datos contiene características y etiquetas que serán utilizadas para entrenar y validar el modelo.

- *Extracción de características y etiquetas*: Se separaron las características (X) y las etiquetas (y) del conjunto de datos. Las características son las variables independientes, mientras que la etiqueta es la variable que queremos predecir.
- *División de Datos*: Se dividió el conjunto de datos en entrenamiento y validación. Esta división asegura que el modelo pueda ser entrenado y validado en diferentes subconjuntos de datos, lo que ayuda a evitar el sobreajuste y proporciona una métrica de rendimiento más realista.
- *Normalización*: Para que todas las características tengan la misma escala y no sesgar el entrenamiento del modelo, se utilizó el *StandardScaler* para normalizar los datos. Esto implica restar la media y dividir por la desviación estándar.
- *Pesos de las clases*: Dado el desequilibrio potencial entre las clases en la variable objetivo, se calcularon los pesos de las clases. Estos pesos se utilizan durante el entrenamiento para dar más importancia a las clases minoritarias, equilibrando así el impacto de cada clase en la función de pérdida.

2) **Construcción del modelo de red neuronal**:
   - *Arquitectura del Modelo*: Se diseñó una arquitectura de red neuronal que consiste en una capa de entrada, capas ocultas y una capa de salida. La capa de salida utiliza una función de activación sigmoide adecuada para tareas de clasificación binaria. Las capas ocultas y el número de neuronas son configurables, permitiendo experimentar con diferentes estructuras.
   - *Envoltura para scikit-learn*: La función `KerasClassifier` se utilizó para envolver el modelo de Keras, permitiendo que el modelo de redes neuronales se integre con las herramientas de `scikit-learn`, en particular con *GridSearchCV* para la optimización de hiperparámetros.
3) **Optimización de hiperparámetros**:
   - *Definición del Espacio de Búsqueda*: Se definió un espacio de hiperparámetros que contiene diversas combinaciones de optimizadores, número de neuronas, número de capas ocultas, funciones de activación, tamaño de los datos de entrenamiento y número de épocas.
   - *Búsqueda Exhaustiva*: Se utilizó *GridSearchCV* para explorar sistemáticamente todas las combinaciones de hiperparámetros en el espacio de búsqueda. Esto asegura que se encuentre la combinación que produce el mejor rendimiento en el conjunto de validación.
   - *Resultados*: Una vez finalizada la búsqueda, se obtuvo la combinación de hiperparámetros que maximiza la precisión del modelo. Esta combinación se puede utilizar para entrenar el modelo final y garantizar su rendimiento óptimo.

Este proceso garantiza que el modelo esté bien sintonizado y optimizado para la tarea en cuestión, aprovechando tanto las capacidades de Keras para la construcción de redes neuronales como las de scikit-learn para la optimización de hiperparámetros.

IV. Conclusion