Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
Inteligencia Artificial II (CI5438)

Redes Neuronales

Estudiantes:

Giacomo Tosone 14-11085

Rommel Llanos 15-10789

Profesor: Carlos Infante

Contenidos

Glosario de Términos:

IT: Iteraciones.
TDE: Tasa de Entrenamiento.
CO1: Capa Oculta 1.
CO2: Capa Oculta 2.
FP: Falsos Positivos.
FN: Falsos Negativos.
CO: Capa Oculta 1.

Sobre el problema

El presente informe aborda dos problemas fundamentales en el campo de la inteligencia artificial y el aprendizaje automático, específicamente utilizando redes neuronales: la clasificación del dataset de Iris y la identificación de mensajes de spam. Estos problemas son emblemáticos en la disciplina, representando desafíos típicos en la clasificación y el procesamiento de datos.

Clasificación del Dataset de Iris

El dataset de Iris es un conjunto de datos clásico en el aprendizaje automático. Compuesto por 150 registros, cada uno con cinco atributos: cuatro características de las flores de iris (longitud y ancho del sépalo, longitud y ancho del pétalo) y la especie de la flor (Iris Setosa, Iris Versicolor, Iris Virginica). El trabajo consiste en desarrollar un modelo de red neuronal que pueda clasificar con precisión estas flores en una de las tres especies mencionadas, basándose en sus características físicas. Este problema es fundamental para entender la clasificación multiclase en redes neuronales y sirve como un banco de pruebas para experimentar con diferentes arquitecturas y parámetros de red.

Clasificación de Spam

La clasificación de spam implica identificar y filtrar correos electrónicos de spam de los mensajes legítimos. Este problema es significativo debido a su aplicabilidad en la vida real y su relevancia en la seguridad de la información. Utilizando un conjunto de datos compuesto por características extraídas de correos electrónicos, el objetivo es entrenar un modelo de red neuronal que pueda predecir con alta precisión si un correo electrónico es spam o no. Este problema presenta desafíos únicos, como el desequilibrio de clases, la necesidad de procesamiento de texto y la importancia de minimizar los falsos positivos (clasificar erróneamente un correo legítimo como spam).

Ambos problemas son esenciales para la comprensión y aplicación de redes neuronales en tareas de clasificación. A través del estudio y experimentación con estos conjuntos de datos, se busca obtener una comprensión más profunda de cómo las variaciones en la arquitectura de la red, los parámetros de entrenamiento y el procesamiento de los datos pueden influir en el rendimiento del modelo y su capacidad para generalizar a partir de datos no vistos anteriormente. La relevancia de estos problemas radica no solo en su aplicación práctica sino también en su capacidad para proporcionar una base sólida en el aprendizaje y la optimización de modelos de inteligencia artificial.

Sobre la instalación

Para el correcto funcionamiento de los scripts de Python utilizados en este proyecto, es esencial seguir el siguiente procedimiento de instalación de los paquetes necesarios:

Python:

- Asegurarse de tener instalada la versión de Python 3.11 o superior.

Instalación de Módulos:

- Pandas: Ejecutar pip3 install pandas para instalar la biblioteca Pandas, utilizada para la manipulación y análisis de datos.
- Matplotlib: Instalar Matplotlib mediante pip3 install matplotlib, necesario para la visualización de datos y gráficos.

Ejecución del Script:

- El script principal, neuralNetwork.py, se encuentra en la carpeta /src.

Sobre los sets de datos

Este proyecto utiliza dos conjuntos de datos principales: el Dataset de Iris y un conjunto de datos para la clasificación de spam. Ambos sets desempeñan roles cruciales en los experimentos de clasificación mediante redes neuronales.

Dataset de Iris

El Dataset de Iris es un conjunto de datos clásico y ampliamente reconocido en el campo del aprendizaje automático. Este dataset contiene 150 observaciones de flores de iris, divididas en tres especies: Iris Setosa, Iris Versicolor e Iris Virginica. Cada observación incluye cuatro características: longitud y ancho del sépalo, y longitud y ancho del pétalo.

Conjunto de Datos de Clasificación de Spam

El conjunto de datos de spam se utiliza para abordar el problema de la clasificación binaria, en particular, para distinguir entre correos electrónicos normales (no spam) y spam. Este conjunto de datos consta de características extraídas de correos electrónicos, incluyendo frecuencias de palabras específicas y otros atributos como la longitud del mensaje y la presencia de ciertos caracteres.

Preprocesamiento y Preparación

En ambos conjuntos el único preprocesamiento de la data que se hizo fue la normalización de los datos, ya que los datos se encontraban limpios y balanceados

Sobre las funciones

Para generar la red neuronal se implementó una clase **NeuralNetwork** con los siguientes métodos que describen el algoritmo de backpropagation

Función forward_propagation:

- Toma inputs como entrada y los pasa a través de las capas de la red.
- Inicializa una lista activations con estos inputs.
- Para cada capa de la red, calcula el producto punto de las activaciones de la capa anterior y los pesos de la capa actual.
- Aplica la función de activación logistica a este resultado para obtener la activación de la capa actual.
- Almacena estas activaciones en la lista activations.
- La función devuelve la lista activations, que contiene las activaciones de todas las capas, incluyendo la salida final de la red.

Función backward_propagation

- Calcula el error en la capa de salida comparando la salida esperada (expected_output) con la activación final de la red (activations[-1]) usando self.lossFn.
- Inicializa una lista errors con el error de la capa de salida.
- Recorre las capas de la red en orden inverso (desde la última hasta la primera capa oculta) para propagar el error hacia atrás.
- En cada capa, calcula el error local utilizando el error de la capa siguiente y los pesos actuales, multiplicado por la derivada de la función de activación (calculada por self.logistic_derivative en las activaciones de la capa).
- Inserta cada error calculado al inicio de la lista errors.
- Devuelve la lista errors, que contiene el error calculado para cada capa de la red.

Función update_weights:

- Recorre cada conjunto de pesos en la red.
- Para cada conjunto de pesos, actualiza los valores aplicando el gradiente del error. Esto se hace mediante el producto punto transpuesto de las activaciones de la capa anterior y el error de la capa actual, multiplicado por la tasa de aprendizaje (learning_rate)

Función train:

- Ejecuta un ciclo de entrenamiento durante un número específico de iteraciones.
- En cada iteración, realiza la propagación hacia adelante de los inputs a través de la red.
- Calcula los errores utilizando la retropropagación, comparando las activaciones resultantes con el expected_output.
- Actualiza los pesos de la red en función de estos errores y la learning_rate dada

Función predict:

- Ejecuta la función forward_propagation con los inputs dados.
- Devuelve la última activación de la lista activations, que representa la predicción final de la red para los datos de entrada proporcionados.

Resultados obtenidos

Parte 2: Iris Dataset

En esta sección se realizó un proceso de pruebas en base al set de datos iris.csv, todo esto con la finalidad de entrenar una red neuronal que permita seleccionar el tipo de planta según sus características, estos tipos de plantas son “Iris Setosa”, “Iris Versicolor” y “Iris Virginica”. Para cada uno de los experimentos se hicieron cambios sobre la data, la tasa de entrenamiento y el número de iteraciones, todo con el fin de entrenar el modelo más apropiado en cada caso estudiado, a continuación en las siguientes secciones se presentan la técnicas utilizadas, así como los resultados obtenidos.

Parte 2.1: Clasificadores binarios

Para esta parte se quiere estudiar cómo los diferentes parámetros de una red neuronal pueden afectar al entrenamiento de una red neuronal cuyo propósito es el poder predecir si una planta pertenece a un tipo específico de especie, para esto se realizaron pruebas sobre qué atributos dan la mejor red neuronal binaria para cada uno de las diferentes especies, es decir se creó al menos una red neuronal por especie.

Otro punto a tener en cuenta es que al crear una red neuronal binaria, el objetivo es solo tener una neurona en la salida, por lo cual la data iris.csv tuvo que ser divida en tres tablas diferentes, donde iriss.csv, irisv.csv y irisvi.csv representan cada una a todos los datos de una especie presentes en iris.csv,, cada uno de estos datos aislados se utilizó como data de entrenamiento de la red neuronal para dicha especie. Por último debido a esta distribución solo se poseen 50 datos por tabla por especie, por lo cual realizar un proceso de validación cruzada sólo dejaría a 40 datos disponibles para el entrenamiento, por lo cual se entrenó a cada uno de los modelos con los 50 datos disponibles para ser luego probado con todos los datos presentes en iris.csv, adicional a esto se realizaron pruebas manteniendo o alterando el orden de los datos de entrenamiento. Los resultados obtenidos por especie son los siguientes:

Parte 2.1.1: Iris Setosa

Para esta parte se realizaron pruebas de diferentes modelos de redes neuronales, para así poder entender cómo los diferentes estados pueden afectar a la red neuronal

IT	TDE	Shuffle	CO1	CO2	FP	FN
5000	0.01	NO	0	0	100	0
5000	0.01	YES	0	0	100	0
50000	0.001	NO	0	0	100	0
50000	0.001	YES	5	0	100	0
50000	0.001	YES	5	10	100	0
50000	0.01	YES	5	10	100	0

Los resultados obtenidos en esta parte no parecen dar información importante, sin embargo esto puede deberse al tamaño de la data o, a la naturaleza de la especie que hace que las posibles aproximaciones fallen. Un dato interesante sacado de este experimento es el cual, los porcentajes de pertenecer a cierta especie se hacen más certeros a medida que se aumentan el número de iteraciones con una tasa de 0.001, adicional a esto agregar una capa oculta de 10 neuronas parece mejorar ligeramente los porcentajes.

Parte 2.1.2: Iris Versicolor

Similar al caso anterior, se quiere entrenar una red neuronal para que sea capaz de predecir si una planta pertenece a la especie Iris Versicolor, se realizaron las pruebas de una manera similar al caso anterior, con la diferencia que se ajustaron de manera diferente, debido a que esta especie parecía responder mejor al modelo de redes neuronales, los resultados obtenidos se encuentran en la siguiente tabla.

IT	TDE	Shuffle	CO	CO2	FP	FN
5000	0.01	NO	0	0	74	0
5000	0.01	YES	0	0	74	0
50000	0.001	NO	0	0	74	0
50000	0.001	YES	0	0	74	0
50000	0.001	YES	5	0	100	0
50000	0.01	YES	50	0	100	0
50000	0.001	YES	5	10	100	0
50000	0.01	YES	5	10	100	0
500000	0.001	YES	5	10	100	0

En base a los resultados obtenidos, se puede denotar que los modelos que mejor resultados ofrecen son los modelos sencillos de una sola capa sin redes ocultas, en particular el modelo entrenado con 50000 iteraciones, una tasa de 0.001 y los datos en el orden original, parece ser el que mejor porcentajes ofrece. En particular para este modelo de datos las capas ocultas pueden desmejorar la efectividad de este modelo.

Parte 2.1.3: Iris Virginica

Similar a los resultados anteriores, este modelo trata de generar una red neuronal capaz de poder clasificar si una especie de planta pertenece a la clase Iris Virginica.

IT	TDE	Shuffle	CO	CO2	FP	FN
5000	0.01	NO	0	0	41	0
5000	0.01	YES	0	0	41	0
50000	0.001	NO	0	0	41	0
50000	0.001	YES	0	0	41	0
50000	0.001	YES	5	0	100	0
50000	0.001	YES	500	0	100	0
50000	0.1	YES	500	0	100	0
50000	0.01	YES	5	10	100	0

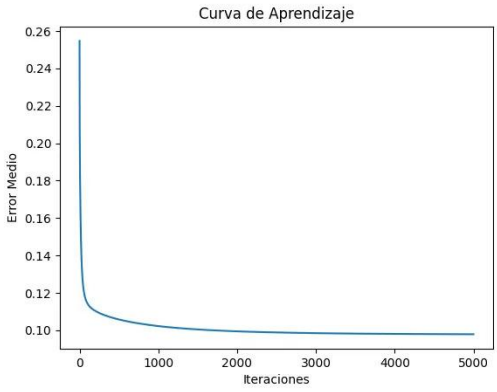
Similar a los casos anteriores, estos modelos parecen ser más eficientes mientras menos capas ocultas poseen, en particular la red neuronal entrenada con 50000 iteraciones, una tasa de 0.001 y los datos en orden aleatorio parecen generar mayores porcentajes. Un dato interesante al realizar experimentos con esta (similar a los casos anteriores) es que parece que agregar capas ocultas disminuye la calidad de este modelo, una hipótesis de la causa de esto puede ser causada por la poca cantidad de datos de entrenamiento, en particular tener solo 50 datos para entrenar los modelos parece ser insuficiente para crear un modelo efectivo.

Parte 2.2: Clasificadores Multiclase

En esta parte se trabajó para crear un modelo de red neuronal que permita clasificar una planta según sus cualidades entre 3 diferentes tipos de especies, adicional a esto se quiere comparar los resultados obtenidos en esta parte con los resultados obtenidos en la parte anterior para definir que método es más efectivo en este caso particular. Similar a casos anteriores, los resultados obtenidos de realizar diferentes experimentos pueden ser revisados en la tabla que se encuentra a continuación.

IT	TDE	Shuffle	CO1	CO2	EP	% Success
5000	0.01	NO	10	10	2	0.9763
5000	0.01	YES	10	10	2	0.9760
5000	0.01	YES	10	0	2	0.9527
50000	0.001	NO	10	0	2	0.9557
50000	0.001	YES	10	0	2	0.9555
50000	0.01	YES	10	10	2	0.9779
50000	0.001	YES	10	10	2	0.9838
50000	0.001	YES	100	10	2	0.9842

50000	0.001	YES	100	100	2	0.9842



TDE: 0.01
CO1: 10
CO2: 10

Los resultados obtenidos por medio de estos diferentes entrenamientos, deja resultados muy interesantes, por un lado parece confirmar la teoría de que la data trabajada parece no acoplarse de manera correcta a una aproximación por redes neuronales, en vista que de manera general los errores no se ven afectados por la escogencia de la tasa, las iteraciones o las capas. El modelo que genera mayores porcentajes de escogencia se encuentra entrenando con 50000 iteraciones, una tasa de 0.001, una capa oculta con 100 neuronas y una segunda capa oculta con 10 neuronas. Existe un caso particular que se genera al realizar este proyecto, a pesar que la cantidad de errores se mantenía constante, las filas que generaban dichos errores si cambian dependiendo del modelo, en particular la fila 83 siempre genera un error pero el segundo error varía en la fila 78 o la fila 132.

Debido a que este modelo parece generar muy pocos errores, es prudente decir que el modelo multivariable es más exacto que la aproximación con clasificados binarios. En particular en el caso de las especies de Iris Setosa en el caso binario no se podía diferenciar de las otras especies, pero el multivariable parece no tener problema en realizar dicha tarea.

Parte 3: Clasificación de Spam

En esta parte del proyecto se quiso estudiar el comportamiento de un modelo de red neuronal frente a una situación en la que se poseyera una mayor cantidad de data, en particular se realizaron diferentes modelos que permiten clasificar (bajo ciertos criterios expuestos en la data) un correo electrónico como spam o no spam. En primer lugar se tuvo que tener cierto grado de familiaridad con la data a trabajar, dicha data se encuentra en el archivo spambase.zip y este comprimible cuenta con 3 archivos, uno es la data en el archivo spambase.data, el segundo archivo spambase.DOCUMENTATION es una explicación de la data y el tercero, el archivo spambase.name presenta todas las columnas de data.

En vista que los nombres de columnas y la data se encuentran en archivos separados, el primer paso es agregar los nombres al archivo spambase.data, en este caso particular se decidió nombrar a las columnas con enteros del 0 al 56 y la última columna se nombró spam ya que dicha columna es usada para la verificación. Esta data presenta la última columna ahora denominada

spam en la que se encuentra la información si el correo electrónico es spam o no, en particular 1 si es spam y 0 si no lo es.

Por último, se decidió realizar dos diferentes tipos de estudios, uno entrenando y probando al modelo con toda la data y el segundo realizando cross-validation con una separación del 80% de la data para entrenamiento y un 20% para pruebas. La razón de esto es que en la data a trabajar se conoce el % de correos spam presentes, si se realizan las pruebas con cross-validation solo se puede observar la cantidad de fallos positivos y negativos presentes. Pero si se trabaja con la data completa se puede observar la cantidad de fallos positivos, negativos y el % de correos de spam predecidos ya que se conoce que en la data a trabajar el % es cercano al 37%.

Parte 3.1: Full Validation

Para este experimento se trata de generar un modelo de red neuronal que permita predecir si un correo es spam, la elección del mejor modelo realizado viene de la mano de dos métricas, una es minimizar la cantidad de fallos positivos/negativos y la otra es tratar de llegar el % de spam en la data lo más cercano a 37% ya que se conoce que dicho % es el % real de la data. Como en partes anteriores, los resultados de los diferentes experimentos se pueden ver en la tabla siguiente:

IT	TDE	CO	FP	FN	% of Spam
5000	0.01	0	267	132	42.3
50000	0.001	0	267	132	42.3
50000	0.01	0	272	120	42.7
50000	0.01	10	28	33	39.29
50000	0.001	10	43	60	39.03
50000	0.01	100	13	11	39.44
5000	0.01	50	63	129	37.97
5000	0.01	100	150	45	41.68

En estos resultados se pueden ver datos interesantes, en primer lugar para este modelo es muy importante una cantidad suficiente de neuronas en la primera capa oculta, se intentó realizar experimentos creando una segunda capa, sin embargo las herramientas de prueba no parecían soportar dicha carga para este modelo. En particular el modelo entrenado con 50000 iteraciones, una tasa de 0.01 y 100 neuronas en la primera capa oculta parece generar los mejores resultados. Para esta data es muy importante la escogencia de una cantidad suficiente de neuronas en la primera capa, debido a esto se necesita una tasa de entrenamiento lo suficientemente alta, de lo contrario se necesitarían más iteraciones para obtener un modelo eficiente.

Parte 3.2: Cross-validation

Una vez estudiado el modelo con la data original, se procedió a entrenar el modelo por medio de cross validation para revisar si los resultados obtenidos por medio del experimento anterior se alinean con los obtenidos por medio de cross validation.

La data fue distribuida de manera aleatoria para cada red neuronal entrenada, por lo cual los porcentajes de Spam encontrados no son un indicador de correctitud en este ejercicio, se tratará por medio de este experimento, la disminución de los fallos positivos/negativos.

IT	TDE	CO	FP	FN	% of Spam
5000	0.01	100	33	42	35.20
50000	0.01	80	45	35	40.20
5000	0.001	100	24	34	40.30
50000	0.01	10	42	45	41.10
50000	0.001	10	35	31	39.80
50000	0.01	0	53	29	42.30
5000	0.01	0	65	31	41.60
50000	0.001	0	48	33	43.10

En vista de estos resultados se puede obtener que para este modelo la escogencia de 100 neuronas en la primera capa oculta es lo más eficiente para este modelo ya que al bajar la cantidad de neurona 80 la cantidad de falsos positivos aumentó, otro dato interesante es que al disminuir un 20% la data de entrenamiento la cantidad de fallos positivos/negativos parece haber aumentado significativamente, ya que a pesar que los números de fallos positivos/negativos parecen similares al experimento anterior, hay que tomar en cuenta que estos datos presentes en la tabla de este experimento son de un espacio de aproximadamente 1000 datos mientras que los del ejemplo anterior son de un universo de 5000 datos. Por lo cual se sigue recomendando para esta data la elección de entrenar el modelo con 50000 iteraciones, una tasa de entrenamiento de 0.01 y una capa oculta con 100 neuronas.