Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus, Querétaro

Inteligencia artificial avanzada para la ciencia de datos I

Módulo 2 Análisis y Reporte sobre el desempeño del modelo.

Nombre y matrícula de la estudiante:
Diego Díaz Ayala A01770236

Profesor:
Benjamín Valdés Aguirre.

Fecha de entrega:
Domingo 10 de septiembre del 2023.

Índice.

Introducción.

1.1 Contexto y descripción de los datos.

Con el fin de seguir aprendiendo diferentes usos para los modelos de machine learning, intente realizar una pequeña implementación de un clasificador de un modelo de red neuronal para que pueda reconocer el sentimiento de una reseña sobre una película.

El modelo busca predecir o categorizar reseñas de diferentes películas para saber si el sentimiento que buscan transmitir es bueno o malo. Este dataset se obtuvo de kaggle en la siguiente liga: https://www.kaggle.com/datasets/atulanandjha/imdb-50k-movie-reviews-test-your-bert

Preprocesamiento de los datos.

2.1 Tokenización y secuencias númericas.

Para la parte de tokenización utilizamos el objeto Tokenizer de la librería de TensorFlow, para poder hacer un modelo de red neuronal con NLP, lo primero que necesitamos hacer es hacer un índice de todas las palabras que están en nuestro dataset.

```
{'<OOV>': 1,
 'the': 2,
 'and': 3,
 'a': 4,
 'of': 5,
 'to': 6,
 'is': 7,
 'br': 8,
 'in': 9,
 'it': 10,
 'i': 11,
 'this': 12,
 'that': 13,
 'was': 14,
 'as': 15,
 'for': 16,
 'with': 17,
```

El parámetro de oov sirve para categorizar todas las palabras que después se mandaran en las secuencias y que no estén en el índice.

Posteriormente tenemos que interpretar cada una de las palabras de nuestro dataset con ese índice.

Y para que podamos meter cada una de las palabras en nuestra capa de entrada en nuestra red neuronal, es necesario hacer un padding a cada una de las oraciones, ya que no todas cuentan con el mismo tamaño. Y nos quedaría una matriz de la siguiente forma.

```
array([[  46,   23,   64, ...,    2, 2721,    5],
       [  48,   24,   57, ...,  683,   15,  109],
       [  11,   67,  114, ...,    1,   11, 8140],
       ...,
       [  12,  492, 1821, ...,    0,    0,    0],
       [  47,  329, 1381, ...,    5, 8586,    3],
       [  83,  123,   11, ...,   23,   68,  189]])
```

Los parámetros que metemos a la función que realiza el padding, son los siguientes:

- Maxlen: Indicamos la longitud máxima que queremos que tengan estas listas.
- Padding: Indicamos donde queremos que se agreguen los 0 (al principio de la lista o final).
- Truncating: Indicamos como queremos que se trunquen los datos en caso de superar la longitud máxima.

Entrenamiento del modelo.

3.1 Configuración del entrenamiento.

La arquitectura que utilice para el modelo de la red neuronal fue el siguiente.

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 100, 16)           160000

 global_average_pooling1d (  (None, 16)                0
 GlobalAveragePooling1D)

 dense (Dense)               (None, 64)                1088

 dropout (Dropout)           (None, 64)                0

 dense_1 (Dense)             (None, 1)                 65

=================================================================
Total params: 161153 (629.50 KB)
Trainable params: 161153 (629.50 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

La capa de embedding se utiliza mucho en modelos de NLP, lo que hace esta capa es que le asigna valores de tipo vectorial a cada palabra, y de esta forma se pueden hacer operaciones mas complejas con estos, y también se manejan en espectros más grandes.

Después hacemos un GlobalAveragePooling1D, lo que hace esta capa, es sacar la media de cada lista, y al final terminamos con vector unidimensional con la media de cada uno.

Las capas ocultas que siguen son simples con una activación ReLU, la capa de dropout sirve para prevenir el overfitting. Nuestra ultima capa tiene una sigmoide como activación para la clasificación entre sus dos clases.

3.1.1 Función de perdida.

La función de perdida que se utilizo para esto modelo fue el Binary Cross Entropy el optimizador que se utilizo fue el Adam.

Evaluación del modelo.

4.1 Resultados en el conjunto de entrenamiento.

Para dividir los datos de entrenamiento entre entrenamiento y validación utilice la función de train_test_split, utilizando un 80% de los datos de entrenamiento para entrenamiento, y el 20% para validación, de esta manera podríamos utilizar todos los datos de prueba para hacerlo, y poder verificar la varianza de nuestro modelo.

Primer modelo de entrenamiento:

Corrimos 30 epochs, y obtuvimos las siguientes medidas

```
Epoch 30/30
625/625 [==============================] - 1s 2ms/step - loss: 0.1131 - accuracy: 0.9812 -
```

4.2 Resultados del conjunto de validación.

Los resultados que obtuve en el conjunto de validación fueron los siguientes.

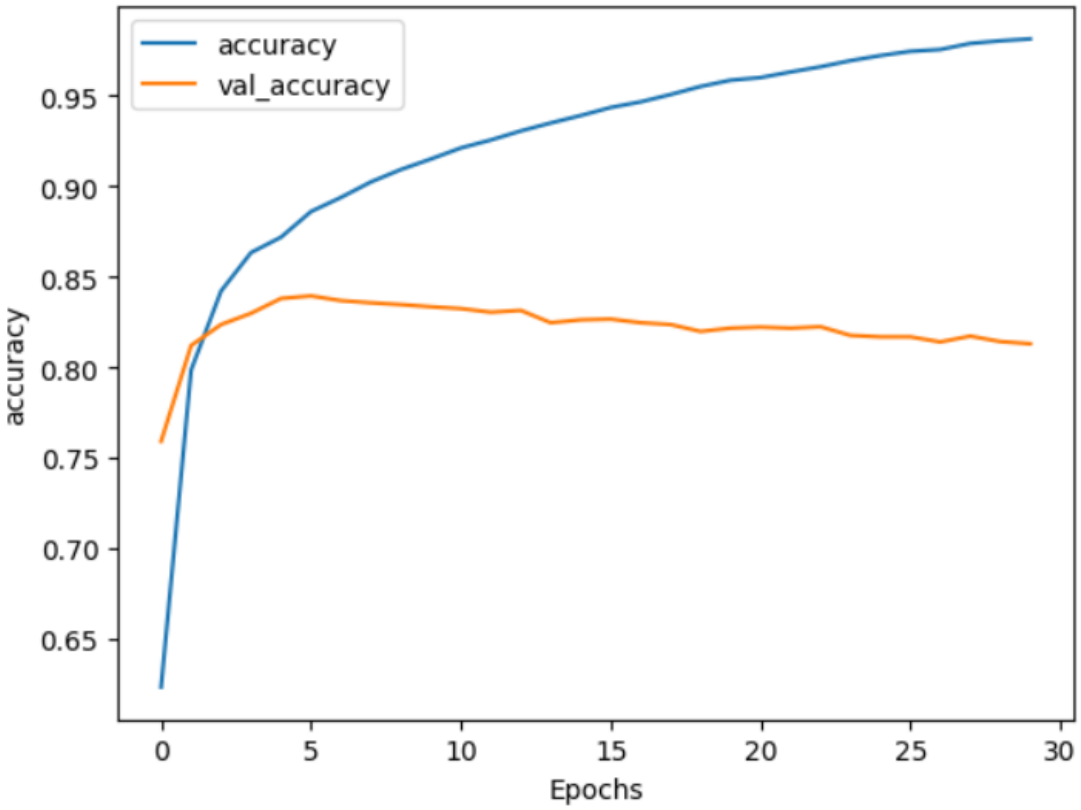
Primera modelo de entrenamiento:

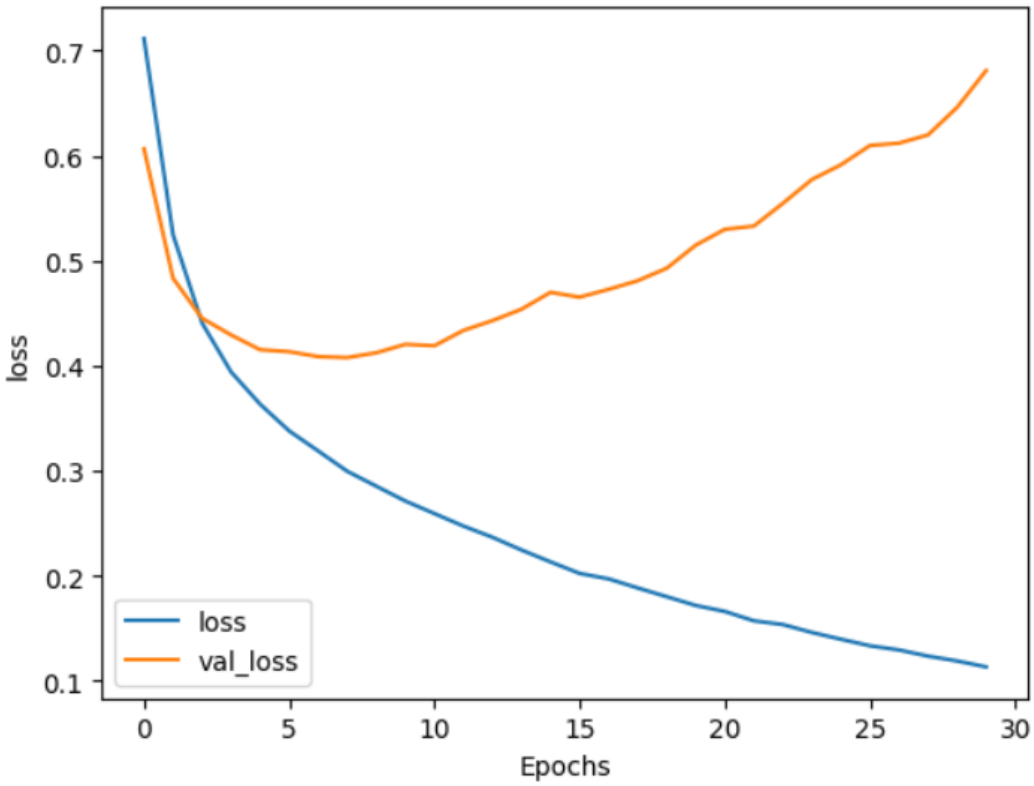
Corrimos 30 epochs, y obtuvimos las siguientes medidas.

```
val_loss: 0.6812 - val_accuracy: 0.8130
```

4.3 Análisis de errores y diagnóstico de nuestro modelo.

Analizando los valores que obtuvimos con nuestro primer modelo corriendo 30 epochs, podemos determinar que nuestro modelo esta teniendo mucho sesgo, ya que la perdida en nuestro conjunto de entrenamiento está bajando, pero la de validación esta subiendo, de igual manera el accuracy en nuestro conjunto de validación está bajando un poco.


accuracy
val_accuracy
accuracy
Epochs
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0
5
10
15
20
25
30


loss
val_loss
loss
Epochs
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
5
10
15
20
25
30

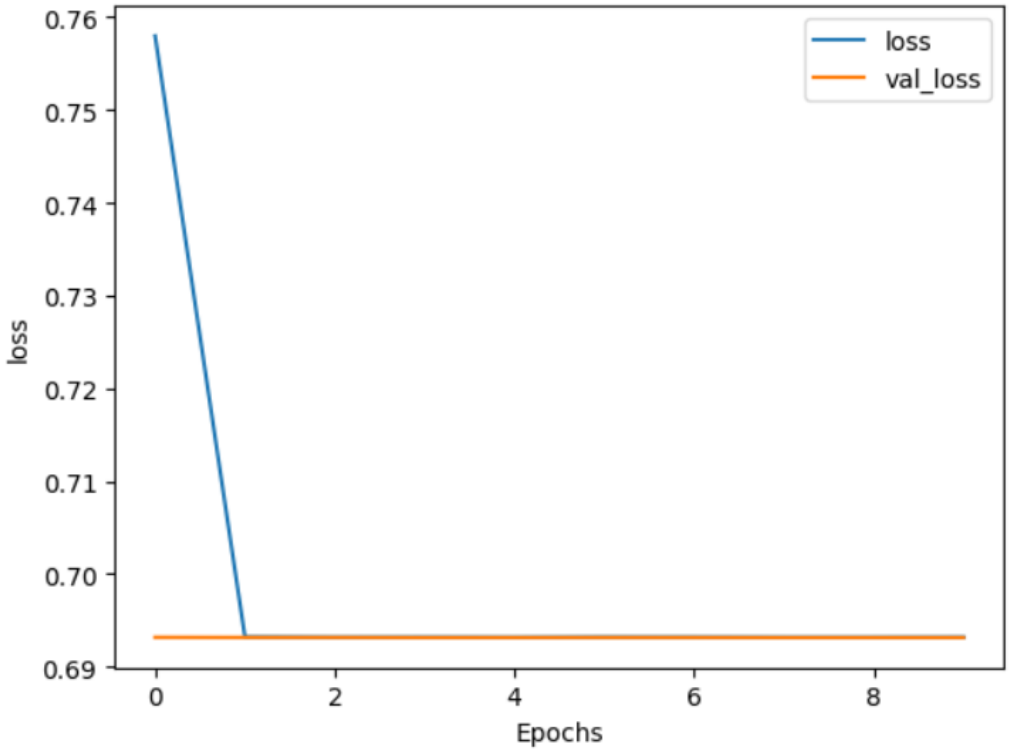
De igual manera nuestro accuracy con nuestro modelo en el conjunto de prueba es el siguiente:

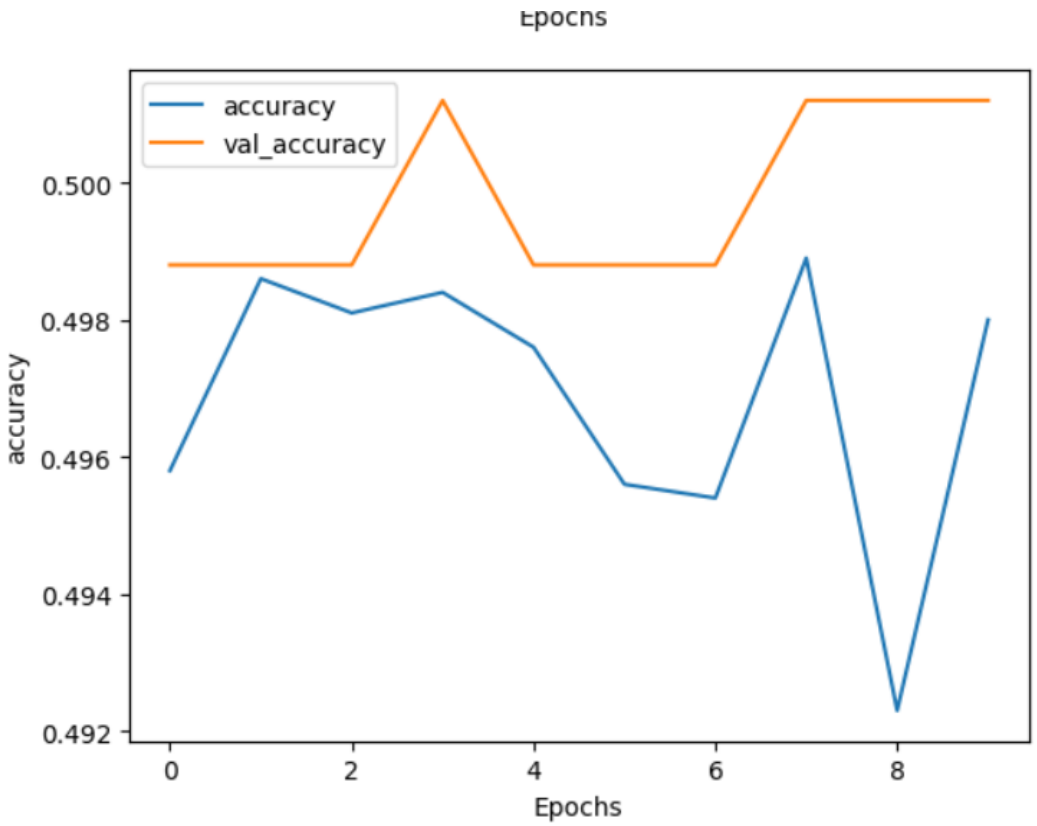
```
782/782 [==============================] - 1s 689us/step
Acurracy:  0.76536
```

Con este análisis lo que podemos determinar es que nuestro modelo está teniendo overfitting.

4.4. Regularización y prevención de overfitting.

Después de intentar implementar mas capas a mi modelo para ver si este requería mas complejidad, me di cuenta de que no era el caso, ya que el accuracy y la perdida no mejoraban con el paso de los epochs.

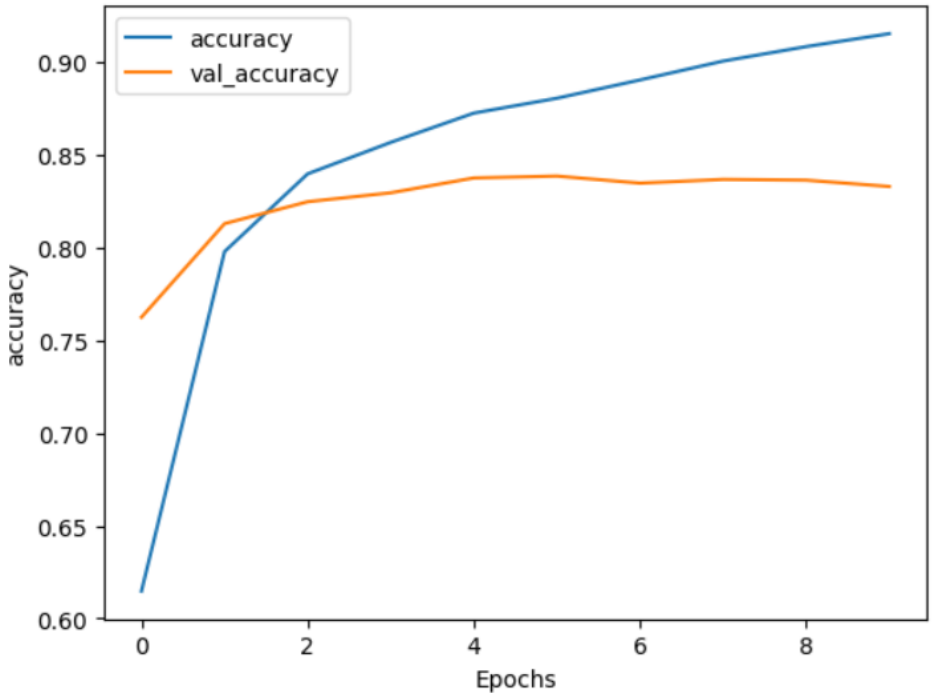


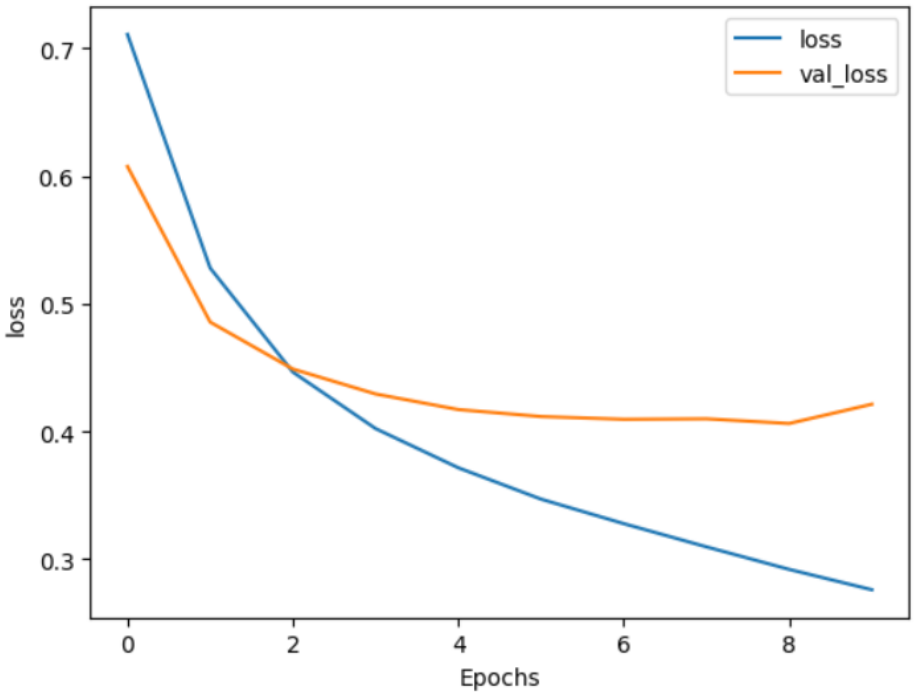


Además de igual forma siguiendo las soluciones para este tipo de casos, un modelo mas complejo no es la solución porque al final lo que se busca es un clasificador binario, entonces opte por mejor reducir el número de epochs a 10.

```
Epoch 10/10
625/625 [==============================] - 1s 2ms/step - loss: 0.2762 - accuracy: 0.9148 - val_loss: 0.4214 - val_accuracy: 0.8326
```

Es el punto en que los valores en el set de validación llegan a su punto máximo.





Y el accuracy en set de prueba mejoro significativamente:

```
782/782 [==============================] - 1s 652us/step
Acurracy:  0.80852
```