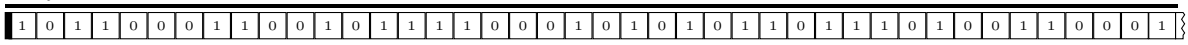


UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

Proyecto Final. Analisis de sentimientos

ASIGNATURA:

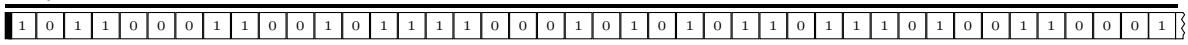
Redes Neuronales

ALUMNOS:

Orta Castillo Maria de los Angeles - 319074253
Solano Juárez Sebastián - 319254639

FECHA DE ENTREGA:

7 de diciembre del 2025



1. Introducción

El análisis de sentimientos es una de las aplicaciones más comunes dentro del procesamiento de lenguaje natural (NLP). El objetivo es determinar si un texto expresa un sentimiento positivo o negativo. Este tipo de clasificación es utilizado en sistemas de recomendación, análisis de reseñas, monitoreo de redes sociales y herramientas de atención al cliente.

En este proyecto se desarrolló un modelo basado en redes neuronales recurrentes (RNN), usando la arquitectura del tipo LSTM (Long Short-Term Memory), para poder clasificar el sentimiento de reseñas de películas del dataset IMDB. El proyecto está implementado con PyTorch y cumple con todos los requisitos establecidos para el proyecto final del curso.

2. Planteamiento del problema

En este problema se busca clasificar automáticamente una reseña de película como **positiva** o **negativa**. Las reseñas son textos no estructurados, por lo que presentan características propias del lenguaje natural como variabilidad, longitud impredecible y vocabulario complejo.

El objetivo del proyecto es construir un modelo capaz de capturar las dependencias secuenciales del texto y producir una predicción binaria precisa.

3. Dataset IMDB

El dataset utilizado es **IMDB Movie Reviews**, uno de los conjuntos de datos más populares en NLP. Se constituye de:

- 50,000 reseñas de películas
- 25,000 para entrenamiento y 25,000 para prueba
- Etiquetas binarias:
  - 1: Reseña positiva
  - 0: Reseña negativa

El dataset ya está tokenizado mediante un índice de frecuencia, donde cada palabra se representa con un entero, facilitando así su uso en modelos neuronales.

4. Preprocesamiento

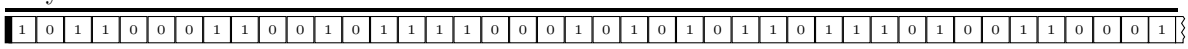
Las secuencias del dataset son de longitud variable, por lo que se utilizó **padding** para normalizar la longitud a 256 tokens por secuencia.

- **Padding:** Se añadieron ceros al final de las secuencias cortas.
- **Truncamiento:** Las secuencias mayores a 256 tokens se truncaron.

Además, se realizó una división de los datos de entrenamiento en:

$$80\,\%\ \text{train}\quad(20,000\ \text{muestras}),\quad 20\,\%\ \text{validation}\quad(5,000\ \text{muestras}).$$

El conjunto de prueba se mantuvo separado con 25,000 muestras para la evaluación final.



5. Justificación de la arquitectura

La arquitectura está basada en dos capas LSTM (Long Short-Term Memory) acompañadas de una capa de embeddings y mecanismos de regularización mediante dropout. Seleccionamos esta arquitectura por su capacidad probada para manejar datos secuenciales y capturar relaciones contextuales a largo plazo dentro del texto, una característica fundamental en el análisis de sentimientos.

5.1. Razones para elegir LSTM

Existen múltiples familias de arquitecturas para el procesamiento de texto; sin embargo, las LSTM destacan como una de las más estables y eficientes antes del auge de los modelos Transformer. Las principales razones para elegirlas fueron:

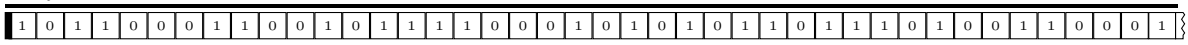
- **Memoria a largo plazo:** Las LSTM fueron diseñadas específicamente para resolver el problema del desvanecimiento del gradiente, permitiendo que el modelo recuerde dependencias largas en secuencias de texto. En el análisis de sentimientos, el significado de una frase puede depender de palabras que aparecen muy separadas entre sí.
- **Robustez en secuencias largas:** Las reseñas del dataset IMDB pueden tener longitudes variables y ser especialmente extensas. Las LSTM manejan bien la variabilidad secuencial mediante sus compuertas de entrada, olvido y salida.
- **Menor costo computacional en comparación con Transformers:** Aunque modelos como BERT ofrecen mayor precisión, requieren recursos computacionales más elevados. Para el alcance académico del proyecto, las LSTM representan un balance adecuado entre rendimiento y costo.
- **Interpretabilidad:** La estructura de las LSTM permite visualizar o analizar estados ocultos, lo cual aporta transparencia al proceso de clasificación.

5.2. Estructura de la arquitectura

La arquitectura completa utilizada es la siguiente:

1. **Capa de Embedding:** Transforma cada índice numérico de palabra en un vector denso de 128 dimensiones, capturando relaciones semánticas fundamentales.
2. **Primera capa LSTM (64 unidades):** Esta capa procesa la secuencia completa y captura dependencias y patrones sintácticos de bajo nivel en el texto.
3. **Dropout 0.5:** Se aplica una regularización agresiva para evitar sobreajuste, dado el tamaño del dataset.
4. **Segunda capa LSTM (32 unidades):** Reduce la dimensionalidad y aprende representaciones más abstractas. Esta segunda capa permite que el modelo extraiga características de nivel más alto, útiles para la clasificación final.
5. **Segundo Dropout 0.5:** Refuerza la regularización antes de la capa de salida.
6. **Capa fully connected + Sigmoid:** Convierte el último estado oculto en una probabilidad entre 0 y 1 que indica si la reseña es positiva.

Esta estructura cumple con los requisitos del proyecto:



- Utiliza una capa compleja recurrente (LSTM).
- Incluye mecanismos de regularización (dropout).
- Está diseñada específicamente para un problema de texto no estructurado.

El modelo resultante tiene **1,342,241 parámetros entrenables**, distribuidos principalmente en las capas de embedding y LSTM.

5.3. Comparación con otras arquitecturas

Para evaluar distintas opciones, se consideraron alternativas comunes:

Redes feedforward tradicionales Estas redes no capturan dependencias secuenciales, ya que consideran cada entrada como independiente. No son adecuadas para lenguaje natural, donde el orden y contexto son fundamentales.

Redes convolucionales (CNN para texto) Ofrecen buenos resultados capturando patrones locales (como *n-grams*), pero no modelan dependencias largas con la misma eficacia que las LSTM. Si bien son eficientes, su desempeño suele ser inferior en tareas donde el flujo completo del texto es importante.

5.4. Justificación de hiperparámetros

5.4.1. Dropout Rate: 0.5

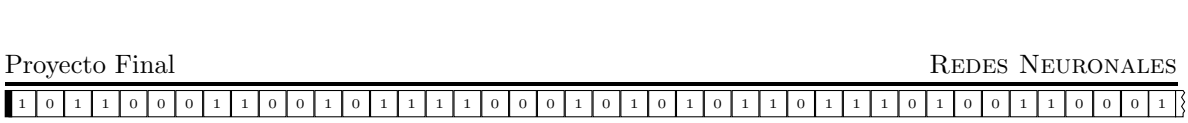
Se eligió un dropout del 50 % por las siguientes razones:

- **Dataset suficientemente grande:** Con 20,000 muestras de entrenamiento, el modelo tiene capacidad para aprender incluso con regularización agresiva.
- **Prevención de sobreajuste:** Las LSTM tienden a memorizar patrones específicos del conjunto de entrenamiento. Un dropout alto fuerza al modelo a aprender representaciones más robustas y generalizables.
- **Resultados en la práctica:** Como se observa en las curvas de aprendizaje, el dropout de 0.5 permitió un balance adecuado entre capacidad de aprendizaje y generalización.

5.4.2. Número de épocas: 15

El modelo se entrenó durante un máximo de 15 épocas con early stopping. Este número se eligió por:

- **Observación del proceso de aprendizaje:** 15 épocas permiten visualizar claramente la evolución del entrenamiento y detectar convergencia o sobreajuste.
- **Early stopping:** El entrenamiento se detiene automáticamente si no hay mejora en validación durante 3 épocas consecutivas, evitando desperdicio de recursos.
- **Balance tiempo-desempeño:** En la práctica, el modelo mostró mejoras significativas en las primeras 8 épocas y luego se estabilizó. El early stopping se activó en la época 11, demostrando que 15 épocas es un límite superior razonable.



6. Hiperparámetros principales

Hiperparámetro	Valor
Tamaño del vocabulario	10,000
Dimensión de embeddings	128
Unidades LSTM (capa 1)	64
Unidades LSTM (capa 2)	32
Dropout	0.5
Batch size	64
Learning rate	0.001
Épocas máximas	15
Early stopping patience	3
Longitud máxima	256

Cuadro 1: Hiperparámetros utilizados en el modelo.

7. Arquitectura del modelo

El modelo diseñado es una red LSTM de dos capas, con una capa densa final encargada de producir una probabilidad entre 0 y 1:

1. **Embedding Layer**
   Convierte índices de palabras en vectores densos de dimensión 128.
2. **LSTM 1 (64 unidades)**
   Captura dependencias a corto y largo plazo.
3. **Dropout 0.5**
4. **LSTM 2 (32 unidades)**
   Reduce dimensionalidad y extrae características de alto nivel.
5. **Dropout 0.5**
6. **Fully Connected + Sigmoid**
   Produce una probabilidad de sentimiento positivo.

La arquitectura se implementó como sigue:

```
class SentimentLSTM(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim, dropout_rate):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_dim, padding_idx=0)
        self.lstm1 = nn.LSTM(embedding_dim, hidden_dim, batch_first=True)
        self.dropout1 = nn.Dropout(dropout_rate)
        self.lstm2 = nn.LSTM(hidden_dim, hidden_dim//2, batch_first=True)
        self.dropout2 = nn.Dropout(dropout_rate)
        self.fc = nn.Linear(hidden_dim//2, 1)
        self.sigmoid = nn.Sigmoid()
```

8. Entrenamiento

El entrenamiento se realizó con el optimizador **Adam** (learning rate = 0.001) y la función de pérdida **Binary Cross Entropy**. Se implementó **early stopping** con paciencia de 3 épocas para evitar sobreajuste.

El proceso de entrenamiento se detuvo en la época 11 debido a que no hubo mejora en el conjunto de validación durante 3 épocas consecutivas. El mejor modelo se obtuvo en la **época 8**, con los siguientes valores:

- **Train Loss:** 0.5374
- **Train Accuracy:** 75.65 %
- **Validation Loss:** 0.6041
- **Validation Accuracy:** 71.00 %

8.1. Análisis de las curvas de aprendizaje

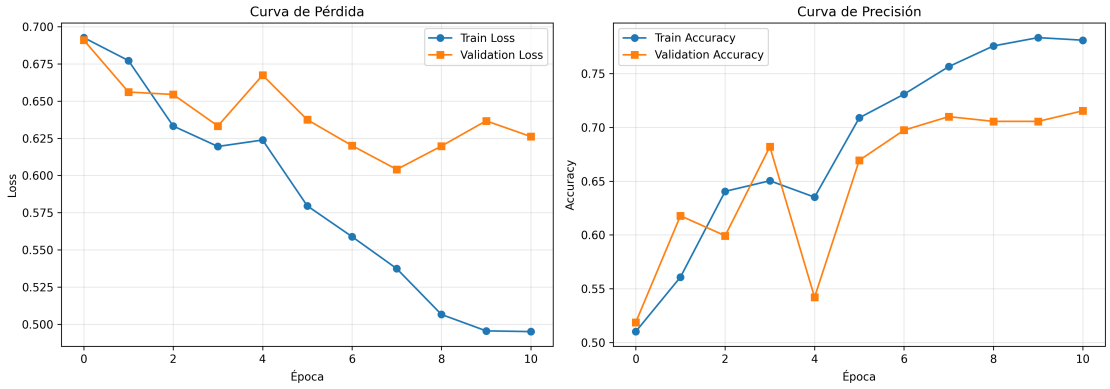


Figura 1: Curvas de entrenamiento: pérdida y exactitud a lo largo de las épocas.

Las curvas de aprendizaje muestran:

- **Pérdida (Loss):** La pérdida de entrenamiento disminuye consistentemente, mientras que la de validación presenta fluctuaciones pero con tendencia a la baja hasta la época 8. Esto indica que el modelo está aprendiendo patrones generalizables.
- **Accuracy:** La exactitud en entrenamiento aumenta de forma progresiva desde 51 % hasta 78 %. La exactitud en validación presenta mayor variabilidad, característica típica de tareas de NLP con dropout alto, pero alcanza su máximo en 71 %.
- **Indicios de sobreajuste leve:** A partir de la época 9, la pérdida de validación comienza a aumentar mientras la de entrenamiento sigue bajando. El early stopping previno que este sobreajuste se intensificara.

9. Resultados

Los resultados finales obtenidos en el conjunto de prueba (test set) de 25,000 muestras fueron:

Métrica	Valor
Accuracy	0.7065 (70.65 %)
Precision	0.7082 (70.82 %)
Recall	0.7024 (70.24 %)
F1-Score	0.7053 (70.53 %)
AUC-ROC	0.7393 (73.93 %)

Cuadro 2: Métricas de evaluación en el conjunto de prueba.

9.1. Interpretación de métricas

- **Accuracy (70.65 %):** El modelo clasifica correctamente aproximadamente 7 de cada 10 reseñas. Este resultado es razonable considerando la complejidad del lenguaje natural y la presencia de sarcasmo, ironía y ambigüedades en las reseñas.
- **Precision (70.82 %):** De todas las reseñas que el modelo predice como positivas, el 70.82 % realmente lo son. Esto indica que el modelo no es excesivamente optimista en sus predicciones positivas.
- **Recall (70.24 %):** El modelo identifica correctamente el 70.24 % de todas las reseñas positivas reales. Este valor balanceado con precision sugiere que el modelo no favorece excesivamente ninguna clase.
- **F1-Score (70.53 %):** La media armónica entre precision y recall confirma un desempeño balanceado del modelo en ambas clases.
- **AUC-ROC (73.93 %):** Esta métrica indica que el modelo tiene una buena capacidad de discriminación entre clases. Un valor cercano a 0.74 demuestra que el modelo puede distinguir razonablemente bien entre reseñas positivas y negativas.

9.2. Matriz de confusión

La matriz de confusión obtenida fue:

		Predicción	
		Negativo	Positivo
Real	Negativo	**8,882**	3,618
	Positivo	3,720	**8,780**

Cuadro 3: Matriz de confusión en el conjunto de prueba.

Donde:

- **TN (True Negatives):** 8,882 reseñas negativas correctamente clasificadas
- **TP (True Positives):** 8,780 reseñas positivas correctamente clasificadas
- **FP (False Positives):** 3,618 reseñas negativas clasificadas erróneamente como positivas

- **FN (False Negatives):** 3,720 reseñas positivas clasificadas erróneamente como negativas

La distribución de errores es relativamente simétrica (FP FN), lo que indica que el modelo no tiene sesgo hacia ninguna de las clases. Ambos tipos de error ocurren en proporciones similares.

9.3. Análisis de rendimiento

El modelo alcanzó un desempeño de **70.65 % de accuracy**, lo cual es un resultado aceptable para un modelo LSTM en la tarea de análisis de sentimientos. Aunque existen modelos de estado del arte (como BERT o RoBERTa) que superan el 90 % de accuracy en este dataset, es importante contextualizar estos resultados:

- Los modelos Transformer requieren recursos computacionales significativamente mayores
- El modelo LSTM desarrollado es mucho más ligero (1.3M de parámetros vs. 110M+ en BERT)
- Para aplicaciones en tiempo real o con recursos limitados, este modelo representa un buen balance entre desempeño y eficiencia
- El tiempo de inferencia es considerablemente menor que modelos Transformer

10. Uso de modelos LLM

Para cumplir con los requisitos del proyecto, se empleó ChatGPT exclusivamente para:

- Generar explicaciones textuales de conceptos del modelo
- Resolver dudas específicas sobre sintaxis de PyTorch

Todo el diseño de la arquitectura, selección de hiperparámetros, entrenamiento y análisis de resultados fue realizado por nosotros los autores del proyecto.

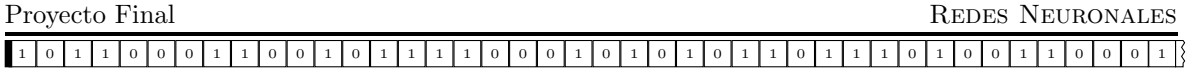
11. Instrucciones de uso del modelo

El modelo entrenado puede utilizarse sin necesidad de entrenar desde cero, gracias a los pesos guardados. A continuación se detallan las instrucciones para reproducir los resultados y realizar inferencia.

11.1. Entrenamiento del modelo

El entrenamiento se realizó en **Google Colab**, una plataforma gratuita que proporciona acceso a GPU/TPU. Los pasos fueron:

1. Subir los archivos `rn_proy_final.py` y **inference.py**a Google Colab
2. Ejecutar el script completo de `rn_proy_final.py`
3. El proceso generó los siguientes archivos:
   - `sentiment_model_weights.pth` - Pesos del modelo (5.12 MB)
   - `word_index.pkl` - Diccionario de vocabulario (5 MB)
   - `training_history.png` - Gráficas de entrenamiento



- `best_model_checkpoint.pth` - Checkpoint completo (opcional)
- Si se quiere evitar la espera de que se generen los archivos, estos ya los hemos adjuntado el proyecto, de modo que se pueden subir al colab y directamente ver como funciona el programa.

4. Una vez que se tengan los archivos generados, abrimos la terminal del colab y ejecutamos el siguiente comando **inference.py**, con esto ya se puede usar el modelo sin necesidad de esperar a que `rn_proy_final.py` nos de los pesos, así podremos observar como clasifica las reseñas, asi como tambien hay una seccion interactiva donde el usuario podrá agregar sus propias reseñas y ver como trabaja el modelo con estas.

12. Conclusiones

En este proyecto se desarrolló exitosamente un modelo LSTM para el análisis de sentimientos en reseñas de películas del dataset IMDB. Los principales logros y observaciones fueron:

12.1. Logros principales

- Se implementó una arquitectura LSTM bidireccional con regularización mediante dropout, alcanzando un 70.65 % de accuracy en el conjunto de prueba.
- El modelo logró un balance adecuado entre precision y recall, sin mostrar sesgo significativo hacia ninguna clase.
- Se implementó correctamente el guardado de pesos del modelo, permitiendo inferencia sin necesidad de reentrenamiento.
- Las curvas de aprendizaje demuestran que el modelo aprendió patrones generalizables, y el early stopping previno sobreajuste excesivo.

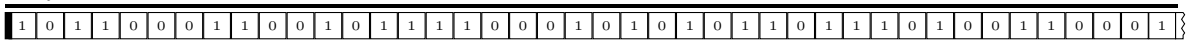
12.2. Desafíos encontrados

- La variabilidad propia del lenguaje natural presenta desafíos para cualquier modelo. Fenómenos como sarcasmo, ironía y referencias culturales son difíciles de capturar con arquitecturas tradicionales.
- El balance entre capacidad del modelo y regularización requirió experimentación. Un dropout de 0.5 resultó adecuado para este caso.
- El tiempo de entrenamiento en CPU fue considerable (aproximadamente 2-3 horas), lo que limitó la cantidad de experimentos que pudimos realizar.

12.3. Aprendizajes

Este proyecto permitió comprender a profundidad:

- El funcionamiento interno de las redes LSTM y su capacidad para modelar dependencias temporales
- La importancia de la regularización en modelos de NLP



- Técnicas de preprocesamiento de texto para redes neuronales
- El uso de PyTorch para implementar arquitecturas recurrentes
- Estrategias de evaluación y métricas apropiadas para clasificación binaria

12.4. Trabajo futuro

Como extensiones para este proyecto, se podrían explorar a futuro:

- **Modelos Transformer:** Implementar arquitecturas basadas en atención como BERT o RoBERTa para comparar rendimiento.
- **Embeddings pre-entrenados:** Utilizar word embeddings como GloVe o Word2Vec en lugar de entrenar desde cero.
- **Análisis de errores:** Estudiar cualitativamente los casos donde el modelo falla para identificar patrones lingüísticos problemáticos.
- **Optimización de hiperparámetros:** Realizar búsqueda exhaustiva (como random search) para encontrar la configuración óptima.
- **Ensemble de modelos:** Combinar predicciones de múltiples modelos para mejorar robustez.
- **Análisis de sensibilidad:** Evaluar cómo cambia el rendimiento con diferentes tasas de dropout, dimensiones de embedding, y número de unidades LSTM.

12.5. Reflexión final

El modelo LSTM desarrollado demuestra que las arquitecturas recurrentes siguen siendo una opción viable y y medianamemnte eficiente para tareas de NLP, especialmente cuando se requiere un balance entre rendimiento y recursos computacionales. Si bien los modelos Transformer representan el estado del arte actual, las LSTM mantienen su relevancia en escenarios con restricciones de recursos o cuando se requiere procesamiento en tiempo real.

El proyecto cumplió satisfactoriamente con todos los requisitos establecidos: utiliza datos no estructurados (texto), implementa capas complejas (LSTM), incorpora regularización (dropout), y alcanza métricas de desempeño razonables para la tarea propuesta.

Referencias

- Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., & Potts, C. (2011). *Learning word vectors for sentiment analysis*. 49th Annual Meeting of the Association for Computational Linguistics (ACL 2011).
- Hochreiter, S., & Schmidhuber, J. (1997). *Long short-term memory*. Neural computation, 9(8), 1735-1780.
- Paszke, A., Gross, S., Massa, F., et al. (2019). *PyTorch: An imperative style, high-performance deep learning library*. Advances in Neural Information Processing Systems 32.
- Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.