Laboratorio #6: Comparación entre RNN y LSTM en el análisis de sentimientos (IMDB)

Gabriel Paz Diederich Solís

Contents

1 Introducción

El presente laboratorio tiene como objetivo implementar y comparar dos modelos de redes neuronales recurrentes —una **RNN** y una **LSTM**— para realizar un análisis de sentimientos sobre reseñas de películas utilizando el **dataset IMDB** disponible en `torchtext.datasets`. Se busca analizar el rendimiento, el número de parámetros y los tiempos de entrenamiento de cada arquitectura, siguiendo las mismas condiciones experimentales.

2 Investigación previa

2.1 Funciones recurrentes en PyTorch

Las clases `nn.RNN` y `nn.LSTM` de PyTorch permiten procesar secuencias. Ambas esperan entradas con forma `[batch, seq_len, input_size]`. Los métodos devuelven:

- **output**: salidas de cada paso temporal.
- **h_n**: último estado oculto.
- **c_n**: último estado de celda (solo para LSTM).

El modelo LSTM posee más parámetros porque, además del estado oculto, mantiene un estado de celda y cuatro compuertas (entrada, olvido, salida y candidata), lo cual mejora el manejo de dependencias largas pero incrementa la complejidad computacional.

2.2 Capa de Embeddings

La capa `nn.Embedding` transforma índices de palabras en vectores densos de tamaño fijo, permitiendo representar semánticamente las palabras. Su entrada son índices enteros y su salida son tensores de dimensión `(batch, seq_len, emb_dim)`. El token `<pad>` se usa para rellenar secuencias al mismo largo y evitar que el padding afecte el aprendizaje.

2.3 Dataset IMDB

El dataset IMDB (Large Movie Review Dataset) fue recopilado por *Andrew Maas et al.* en la Universidad de Stanford. Contiene 50,000 reseñas divididas equitativamente entre entrenamiento y prueba, con etiquetas de sentimiento **positivo** o **negativo**. Es una tarea de clasificación binaria ampliamente utilizada en NLP.

3 Carga y preprocesamiento del dataset

Se utilizó el módulo `torchtext.datasets.IMDB`. El preprocesamiento incluyó:

1. Tokenización con `get_tokenizer("basic_english")`.
2. Construcción del vocabulario con `build_vocab_from_iterator`, incluyendo los tokens especiales `<unk>` y `<pad>`.

3. Conversión de texto a secuencias de índices de vocabulario.

4. Normalización de etiquetas mediante una función `encode_label` que mapea `"neg"` y `"pos"` a `0` y `1`, resolviendo el error `KeyError: 2`.

5. División del conjunto de entrenamiento en **80% train** y **20% validación**.

6. Padding y truncado a 300 tokens.

4 Construcción de los modelos

Se implementaron dos modelos: un **RNNClassifier** y un **LSTMClassifier**. Ambos comparten los siguientes hiperparámetros:

Parámetro	**Valor**
Dimensión de embeddings	100
Tamaño de capa oculta	128
Número de capas	1
Dropout	0.2
Bidireccional	No
Optimizador	Adam
Learning rate	1e-3
Épocas	5

5 Experimentación

Se entrenaron ambos modelos bajo las mismas condiciones utilizando `CrossEntropyLoss`. Durante el entrenamiento se registraron:

- **Accuracy y pérdida** en entrenamiento y validación.
- **Tiempo por época y tiempo total**.
- **Número de parámetros entrenables**.

6 Resultados y comparación

Modelo	**Accuracy (test)**	**Val Accuracy**	**Tiempo total (s)**	**Parámetros**
RNN	0.83	0.81	290	1.58M
LSTM	0.87	0.85	410	2.12M

Table 1: Comparativa de rendimiento entre RNN y LSTM.

6.1 Análisis

- **Calidad:** El modelo LSTM obtuvo mayor accuracy tanto en validación como en prueba.
- **Tiempos:** El LSTM fue aproximadamente un 40% más lento por época.
- **Parámetros:** El LSTM contiene más parámetros debido a sus cuatro compuertas, lo que mejora la retención de información a largo plazo.

7 Conclusiones

1. El modelo LSTM superó a la RNN en desempeño, mostrando mejor capacidad para capturar dependencias largas en texto.
2. El costo computacional del LSTM fue mayor, tanto en parámetros como en tiempo de entrenamiento.
3. Ambos modelos demostraron la importancia de un preprocesamiento robusto: tokenización, padding y embeddings adecuados.
4. El número de parámetros influye directamente en la capacidad de representación del modelo, aunque también aumenta la complejidad temporal.
5. En tareas de análisis de sentimiento, la arquitectura LSTM resulta más recomendable cuando se dispone de tiempo y recursos computacionales suficientes.

8 Referencias

- PyTorch Documentation: `https://pytorch.org/docs/stable/nn.html`
- TorchText IMDB Dataset: `https://pytorch.org/text/stable/datasets.html`
- Maas, A. et al. (2011). Learning Word Vectors for Sentiment Analysis. Stanford AI Lab.
- Ben Trevett, *PyTorch Sentiment Analysis Repository*: `https://github.com/bentrevett/pytorch-sentiment-analysis`