Tarea 1. Aplicación de RNNs a la tarea del modelamiento de lenguaje
Curso: Fundamentos de Computación inteligente

Objetivo

El objetivo de esta tarea es experimentar con modelos de Redes Neuronales Recurrentes (RNNs), específicamente LSTM (Long Short-Term Memory) y BiLSTM (Bidirectional LSTM), para realizar el modelado del lenguaje en español. El modelo debe predecir la palabra siguiente dada una secuencia de palabras (oración o sentencia), utilizando técnicas modernas de procesamiento del lenguaje natural (NLP).

Este ejercicio permite explorar cómo los modelos de etiquetado secuencial aprenden patrones lingüísticos, como dependencias temporales, contexto y semántica, a partir de datos textuales. Para entrenar los modelos tenemos los siguientes componentes:

Componentes del proyecto

1. Carga del Dataset

- Se utilizará el dataset spanish_billion_words_clean disponible en Hugging Face.
- Este corpus contiene millones de oraciones en español, lo que permite entrenar modelos robustos.
- Se puede limitar el tamaño del dataset para experimentos iniciales (ej: 5000–50000 oraciones).

```
from datasets import load_dataset

dataset = load_dataset("text", data_dir="path/to/spanish_billion_words_clean")
sentences = dataset["train"][:50000]  # Limitar tamaño
```

2. Tokenización y Creación del Vocabulario

1) Se aplicará tokenización de palabras usando Tokenizer de Keras.
2) Se generará un vocabulario de palabras únicas.
3) Cada palabra se mapea a un índice entero (word_index).
4) Se crea un diccionario inverso (index_word) para reconstruir texto.

tokenizer = Tokenizer(oov_token="<OOV>")

tokenizer.fit_on_texts(sentences)

sequences = tokenizer.texts_to_sequences(sentences)

vocab_size = len(tokenizer.word_index) + 1

3. Creación del Conjunto de Entrenamiento (X, Y)

- X: Secuencias de entrada (lista de listas de índices)
- Y: Índice de la palabra siguiente (**no one-hot**, ya que usamos `sparse_categorical_crossentropy`)
- Para cada secuencia `[w1, w2, ..., wn]`, se crean pares:
  - `(w1) → w2`
  - `(w1, w2) → w3`
  - ...
  - `(w1, ..., wn-1) → wn`

4. Padding y Truncado

- Se calcula `MAX_LEN = max(len(x) for x in X)`
- Se aplica padding (`pad_sequences`) para alinear todas las secuencias a `MAX_LEN`.
- Se puede truncar si `MAX_LEN` es muy alto (> 50) para mejorar eficiencia.

MAX_LEN = min(50, max(len(x) for x in X))

X_padded = pad_sequences(X, maxlen=MAX_LEN, padding='pre')

5. División del Conjunto (Train/Test)

- División en 80% entrenamiento / 20% prueba.
- Uso de `train_test_split`:

6. Construcción del Modelo de LSTM o BiLSTM

Construcción del modelo secuencia del LSTM, para ello podemos usar una dimensión de embedding de 100 (o más hasta 300) y 64 unidades para LSTMs aunque podemos probar LSTMs de 128 unidades. La función de activación ReLU (Rectified Linear Unit) es rápida y no satura como las funciones **sigmoide y tanh.** La Dense(64) con la ReLU reducen gradualmente la dimensionalidad y permiten actuar en un espacio más "pequeño". Esta función de activación toma la salida de la capa anterior de 64 o 128 y le aplica una transformación lineal y produce un vector de 64 valores que van a la capa final Dense(vocab_size, softmax). La creación del modelo también tiene el modo de compilación, en este caso usamos la función de pérdida de **sparse_categorical_crossentropy** que no necesita one-hot para la Y, también usaremos el optimizador de **adam**.

7. Entrenamiento del modelo con Early Stopping

El entrenamiento del dataset (X,Y), en la que se tienen en cuenta las épocas de entrenamiento y el batch_size que es el tamaño de las muestras. Para evitar el overfitting o sobreajuste se debe definir la técnica de Early stopping para la prevención del overfitting el cuál detiene el entrenamiento del modelo antes que comience a sobreajustarse.

```
early_stopping = EarlyStopping(
    monitor='val_loss',           # ← Métrica que vigila
    patience=15,                  # ← Número de épocas sin mejora antes de parar
    restore_best_weights=True,    # ← Recupera los pesos del mejor modelo
    verbose=1                     # ← Muestra mensaje cuando se detiene
)
EPOCHS = 30
BATCH_SIZE = 128

print("\nEntrenando modelo BiLSTM...")
history = model.fit(
    X_train, y_train,  # <-- etiquetas como enteros
    epochs=EPOCHS,
    batch_size=BATCH_SIZE,
    validation_split=0.1,
    callbacks=[early_stopping],    # ← Aquí va el callback
    verbose=1
)
```

8. Cálculo de la perplejidad

El cálculo de la perplejidad del conjunto de testeo es una de las métricas más importantes en modelamiento del lenguaje, para la predicción de la palabra siguiente. Si el modelo siempre está muy seguro y acierta fácilmente → perplejidad baja → ¡buen modelo. Si el modelo siempre está confundido y le parece que cualquier palabra podría venir → perplejidad alta → ¡mal modelo!. *En términos generales, la perplejidad mide la sorpresa del modelo ante la palabra correcta. Para cada palabra real, se toma la probabilidad que el modelo le asignó*

(de entre todo el vocabulario), se calcula el logaritmo para suavizarla, se promedia sobre todo el conjunto de prueba, y finalmente la perplejidad es el exponencial del negativo de ese promedio.

- $P(w_t | w_{1:t-1})$ = probabilidad que el modelo asigna a la palabra correcta w_t en la posición t

Entonces:

$$\text{Perplejidad} = \exp\left(-\frac{1}{N}\sum_{t=1}^{N}\log P(w_t | w_{1:t-1})\right)$$

Donde N es el número total de palabras del conjunto de testeo. La siguiente es la implementación de la perplejidad del conjunto de testeo.

```python
import numpy as np

def calculate_perplexity(model, X_test, y_test, tokenizer):
    probs = model.predict(X_test, verbose=0)
    log_probs = np.log(probs[np.arange(len(y_test)), y_test])
    avg_log_prob = np.mean(log_probs)
    perplexity = np.exp(-avg_log_prob)
    return perplexity
```

Estos son algunos valores de perplejidad con su descripción.

VALOR DE PERPLEJIDAD	¿QUÉ SIGNIFICA?
1	Modelo perfecto (imposible). Siempre asigna probabilidad 1 a la palabra correcta.
5 - 20	Modelo excelente (como los grandes LLMs en sus idiomas principales).
20 - 100	Modelo bueno (típico en modelos pequeños bien entrenados).
100 - 500	Modelo aceptable, pero con margen de mejora.
500 - 1000+	Modelo pobre: está muy "perplejo", no sabe qué palabra sigue.

9. Predicción de la próxima palabra

Predicción de la próxima palabra dada una sentencia X dada por el usuario, en este caso se crea un procedimiento para predecir la próxima palabra **predict_next_word(model, tokenizer, user_input, MAX_LEN)** que tenga en cuenta el modelo , la tokenización y la enterización de la sentencia de entrada.

Para la entrega de los resultados se deben tener en cuenta la información de las siguientes tablas:

Tabla1 Resultados esperados

Modelo	**No sentencias-dataset**	**Perplejidad**	**Observaciones**
LSTMs	5000 -20000 -50000	< 100 es muy buen resultado y >500 ; hay que usar más épocas, usar más datos, aumentar el embedding del LSTM (100-300) y también la longitud de las secuencias, es decir aumentar el MAX_LEN	- Baja perplejidad → modelo seguro y acierto alto → buen trabajo! - Alta perplejidad → modelo confundido → necesita más datos, más épocas o ajustes.
BiLSTMs	5000- 20000- 50000	Similar o mejor que LSTM	- Capta contexto bidireccional → mejor rendimiento en tareas complejas. - Requiere más recursos computacionales.

Tabla 2. Parámetros de todo el proceso de predicción

Parámetros	Recomendación
Tamaño del dataset	Usar más datos (≥ 50,000 oraciones) para mejorar generalización
Dimensión del embedding	Probar entre 100 y 300 (mejor para capturar semántica)
MAX_LEN	Ajustar según longitud promedio de oraciones (ej: 40–60)
Unidades LSTM	Probar 64, 128, 256; 64 es suficiente para datos pequeños
Dropout	Añadir dropout=0.2 para evitar overfitting
Batch size	32, 64, 128
Épocas	30, 50 , 80 (dependiendo del tamaño del dataset) Cuando se tienen pocos datos en el dataset es mejor repetirlos con más épocas. Se entendería que con más datos el modelo aprende más rápido y por ello es mejor usar

Generalidades de entrega

La tarea será realizada en grupos de máximo tres estudiantes. Se debe entregar el archivo fuente en en ****.ipynb el cuál debe estar lo mejor documentado. El plazo del trabajo son dos semanas. Se puede realizar con Jupyter AI o en colab. Para la calificación de la tarea se tendrán en cuenta la información de la tabla 1 y también el uso de los parámetros y recomendaciones de la tabla 2.