TEMA 4 – HOJA DE EJERCICIOS II

Se plantean una serie de ejercicios para practicar con las redes neuronales recurrentes vistas en clase.

Nota: Se debe de consultar el api de Keras para ver los diferentes parámetros de configuración de las redes a utilizar.

Ejercicio 1 - RNN

Se propone entrenar modelos simples de redes neuronales recurrentes (RNN por sus siglas en inglés) sobre un conjunto de noticias clasificadas como verdaderas o falsas (fake news).

Para ello, se utilizará el ficherto **train.xlsx** del conjunto de datos The *Spanish Fake News Corpus,* visto en ejercicios anteriores. Puedes obtener de nuevo dicho fichero a través del siguiente enlace:

https://github.com/jpposadas/FakeNewsCorpusSpanish/tree/master

1) **Carga del dataset -** Descarga el anterior dataset y obtén los títulos de las noticias (columna *Headline*), el contenido de estas (columna *Text*) y sus etiquetas (columna *Category*). Clasifica las etiquetas en 0 si la noticia es real (*True*) y 1 si es falsa (*Fake*).

2) **Análisis exploratorio -** Analiza la distribución del número de palabras tanto en los títulos de las noticias (frases cortas) como en su contenido (varias frases). Este análisis será utilizado posteriormente para decidir el tamaño máximo a introducir por cada documento.

3) **Separación del dataset -** Divide los títulos de las noticias y su contenido en sus correspondientes conjuntos de entrenamiento, validación y test. Recuerda que los conjuntos de entrenamiento serán usados posteriormente tanto para la construcción de los vocabularios como para el entrenamiento de las redes neuronales.

4) **Construcción del vocabulario -** Construye los vocabularios tanto para los títulos de las noticias como para su contenido. Realiza un preprocesamiento básico de las palabras que los componen.

5) **Codificación de los textos** - Construye las matrices de documentos-términos, donde cada palabra es representada por su índice en el vocabulario correspondiente. Para la elección del tamaño máximo, usa el análisis anteriormente realizado sobre la distribución del número de palabras.

6) **Entrenamiento sobre los títulos**

a. Empleando la capa SimpleRNN() de Keras, diseña y entrena una red neuronal recurrente para los títulos de las noticias. Entrena la capa de embeddings desde cero. Usa los correspondientes conjuntos de validación anteriormente creados para evaluar cómo se van entrenando los modelos diseñados.
b. Prueba distintas configuraciones de la red neuronal (número de neuronas, número de dimensiones de la capa de Embeddings, Dropout / Recurrent Dropout…).
c. Con los modelos entrenados sobre los títulos de las noticias, realiza predicciones sobre su correspondiente conjunto de test y analiza los resultados obtenidos.

7) **Entrenamiento sobre los contenidos**

a. Repite el proceso anterior de entrenamiento de redes neuronales recurrentes para el contenido de las noticias usando la capa SimpleRNN() de Keras.
b. Nuevamente, experimenta sobre distintas configuraciones y diseños de entrenamiento.
c. Con los modelos de redes neuronales entrenados sobre el contenido de las noticias, realiza predicciones sobre su correspondiente conjunto de test. Analiza los resultados obtenidos y compáralos sobre los anteriormente obtenidos con los títulos.

8) **Entrenamiento con embeddings de Word2Vec**

a. En vez de entrenar los embeddings desde cero, emplea un modelo de Word2Vec en español para su inicialización. Puedes usar el modelo “SBW-vectors-300-min5.txt” visto en ejercicios anteriores (https://www.kaggle.com/datasets/rtatman/pretrained-word-vectors-for-spanish).
b. Inicializa los embeddings de las redes neuronales diseñadas anteriormente usando Word2Vec. Prueba distintas configuraciones de redes neuronales y analiza los resultados obtenidos.

Ejercicio 2 - LSTM

En este ejercicio se propone entrenar redes neuronales recurrentes con memoria a largo y corto plazo (LSTMs por sus siglas en inglés) sobre el mismo conjunto de datos.

- Repite los pasos realizados en el Ejercicio 1 tanto para los títulos de las noticias como para su contenido completo, empleando en esta ocasión la capa **`LSTM()`** de la librería Keras.
- Experimenta nuevamente distintas configuraciones de la red neuronal, modificando sus hiperparámetros (número de unidades LSTM, dimensiones de la capa de Embeddings, regularización con Dropout o Recurrent Dropout, etc). Prueba a inicializar los embeddings de forma aleatoria (entrenamiento desde cero) o por medio del uso del modelo de Word2Vec "SBW-vectors-300-min5.txt".
- Analiza los resultados, comparando las diferentes configuraciones de LSTM experimentadas y los resultados obtenidos durante el Ejercicio 1.

Ejercicio 3 – Bi-LSTM

En este ejercicio se propone entrenar redes neuronales recurrentes LSTM Bidireccionales sobre el mismo conjunto de datos.

Para configurar la red se pueden añadir tantas capas como quieras, pero como siempre, cuánto más compleja sea la red mayor será su capacidad para aprender la mejor representación, pero también aumentará notablemente la complejidad temporal de su proceso de entrenamiento. ¿Qué se puede hacer para determinar el número óptimo de capas (y células en cada capa)? Pues lo razonable es hacer varios experimentos.

A continuación, proporcionamos una configuración de partida:

- Entre capa y capa se va a utilizar una capa dropout (técnica de regularización para reducir el sobre-ajuste). En esta capa se omiten algunas neuronas durante el entrenamiento (estas neuronas se seleccionan aleatoriamente en cada ciclo). Vamos a usar una probabilidad de 0.4 => cada 10 neuronas 4 serán omitidas.

  Después de las capas BiLSTM, también se añade una capa densa.

```
from keras.layers import Bidirectional, LSTM, Dropout, Dense

DROPOUT = 0.4

model.add(Bidirectional(LSTM(60, return_sequences=True, recurrent_dropout=0.2)))
# Añadimos una capaz droput después de la capa bilstm
model.add(Dropout(DROPOUT))

model.add(Bidirectional(LSTM(32, recurrent_dropout=0.2)))
model.add(Dropout(DROPOUT))

model.add(Dense(60, activation='relu'))
```

- La última capa usa la función sigmoid y devuelve una probabilidad. Si la probabilidad es cercana a 1, la capa devuelve 1, y 0 en otro caso.

```
model.add(Dense(50, activation='relu'))
# Para clasificación binaria
model.add(Dense(1, activation='sigmoid'))
```