Universidad Rey Juan Carlos

TEMA 4 – HOJA DE EJERCICIOS I

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

Ejercicio 1

Se propone un ejercicio para ver cómo entrenar un modelo simple de red neuronal utilizando la biblioteca Keras de Tensorflow. Se va a utilizar un conjunto muy básico de frases en español para clasificarlas según su sentimiento (positivo o negativo).

El conjunto de frases con su anotación es el siguiente:

- "Estoy un poco harto del día a día, nada mejora" -> Negativo
- "Hoy es un buen día" -> Positivo
- "No se te ve satisfecho con el trabajo" -> Negativo
- "Este paisaje es hermoso y bonito" -> Positivo

1) En este primer ejercicio no se va a realizar ningún tipo de preprocesamiento del texto. Las etiquetas (`labels`) serán 1 para la clase positiva y 0 para la clase negativa. Se define una longitud para los textos = 10. Se define un tamaño de vocabulario = 50.

Hay que ir siguiendo los pasos que se indican a continuación:

2) Construir el vocabulario (los tokens diferentes de los textos de entrada). Mostrar los tokens del vocabulario con su índice.
3) Construir la matriz de documentos-términos con el tamaño máximo elegido, donde el peso de cada término es el índice en el vocabulario (si no está se le pone valor 0).
4) Una vez preprocesado el texto, hay que definir el modelo. Se va a utilizar la clase `Sequential`, que permite añadir diferentes capas de forma secuencial.
5) La primera capa del modelo es la capa de embeddings, que se encargará de representar cada texto como una matriz de vectores. La matriz tendrá como número de filas el valor de la variable `max_length`, es decir, la longitud de las secuencias de entrada. Como número de columnas, tendremos que decidir que dimensión queremos utilizar para representar los vectores de los tokens. En este ejemplo, vamos a utilizar un tamaño de vector pequeño (`vector_size` = 8),

porque estamos con un ejemplo de juguete (las dimensiones que se suelen utilizar más son 100, 200 o 300).

La capa Embedding lo que hará será inicializar una matriz por cada texto. Como se ha dicho antes la matriz, tendrá una dimensión de `max_length x vector_size`.

En este ejercicio, la matriz se inicializa con pesos aleatorios, pero se podría inicializar la matriz a partir de un modelo pre-entrenado de word embeddings (lo veremos en otro ejercicio).

Hay que añadir una capa densa para la salida. En este caso, al ser una salida binaria como función de activación podemos utilizar sigmoid. Devolverá una probabilidad, que si es cercana a 1 entonces la capa de salida devolverá 1, 0 en otro caso.

6) Compilar el modelo, al hacer esto hay que indicar el optimizador de pesos a utilizar, la función de pérdida/error y las métricas que se van a utilizar en cada epoch.

- Optimizador (optimizer): encargado de cambiar los atributos de la red neuronal como los pesos y la tasa de aprendizaje para reducir las pérdidas (e.g. 'adam', o 'rmsprop').
- Función de pérdida (loss): mide cómo de bien el modelo está haciendo sus predicciones comparadas con los valores reales. El objetivo del entrenamiento es minimizar esta función.
- Evaluación (metrics): métricas utilizadas para evaluar el rendimiento del modelo. No se utilizan para entrenar el modelo, pero son importantes para analizar cómo está funcionando el modelo.

7) Una vez que se ha configurado y compilado el modelo, se puede entrenar. Para ello, simplemente hay que llamar al método `fit`. Este método recibe los siguientes argumentos:

- Los textos preprocesados y almacenados en la variable `encoded_train_pad`, y sus `labels` correspondientes, que el modelo utilizará para medir el error y aprender.
- El número de epochs y el tamaño del batch. En este ejercicio, utilizaremos 5 epochs y un tamaño de batch de 32. Aunque lo razonable es probar con diferentes valores para ver cuáles proporcionan los mejores resultados.
- Los datos de validación. Si no se tiene conjunto de validación explícito, en el método `fit` (para entrenar el modelo) se puede especificar qué porcentaje del training se usaría como conjunto de validación. Para ello se usa el argumento `validation_split`.

- Además, también vamos se podría usar `EarlyStopping` que permite parar el entrenamiento si después de un número *n* de epochs (patience), el error sobre el conjunto de validación no ha disminuido.

8) Por último, se valida el modelo con un conjunto de frases de prueba:

   - "No fui al estreno de la película porque nadie me quería acompañar"
   - "Envidio de buena manera a los que tienen la oportunidad de ir mañana al estadio"
   - "Se nos está volviendo costumbre del domingo por la noche, ver el episodio anterior de SNL y eso me hace recibir el lunes con mejor humor"
   - "Al final decidí no ir al cine porque estaba cansada"

Nota: al tener una inicialización de pesos aleatoria, si se ejecuta varias veces sin cambiar la configuración, los resultados pueden ser distintos.

Ejercicio 2

Dado que en el ejercicio anterior el conjunto de entrada era muy pequeño, se quiere poder entrenar un modelo con más datos. Para ello, se propone utilizar el conjunto https://huggingface.co/datasets/cardiffnlp/tweet_sentiment_multilingual. Este conjunto tiene mensajes de Twitter en 8 idiomas diferentes.

Cada instancia en este conjunto de datos tiene esta forma:

{'label': 2, 'text': '"QT @user In the original draft of the 7th book, Remus Lupin survived the Battle of Hogwarts. #HappyBirthdayRemusLupin"'}

La etiqueta puede ser 0 para sentimiento negativo, 1 para sentimiento neutro y 2 para positivo. En este ejercicio seguiremos planteando una clasificación binaria (1 positivo y 0 negativo). Por lo tanto, la etiqueta 2 será 1 para nuestro modelo y las etiquetas 1 y 0 serán 0 para nuestro modelo.

Para este ejercicio elegiremos solo textos en español y definimos un tamaño de vocabulario de 5000 y una longitud máxima de texto de 100. A continuación, se muestra el preprocesamiento que se puede hacer para estos datos, para ajustar las labels a las indicadas y la configuración de tamaños también.

```python
from datasets import load_dataset

# Cargar el dataset desde Hugging Face
dataset = load_dataset("cardiffnlp/tweet_sentiment_multilingual", "spanish", split='train')

# Inspeccionar columnas
print(dataset.column_names)

# Filtrar datos para solo etiquetas positivas y negativas
filtered_data = dataset.filter(lambda x: x['label'] in [0, 2])


# Extraer texto y convertir etiquetas a binario
labels = [1 if label == 2 else 0 for label in filtered_data['label']]
texts = filtered_data['text']
vocab_size = 5000
max_length = 100
prepared_sentences, vocabulary = prepare(texts, vocab_size, max_length)
print(f"Tamaño total del dataset: {len(texts)}")
print(f"Distribución de etiquetas: Positivos={labels.count(1)}, Negativos={labels.count(0)}")
```

Se pide utilizar una CNN para clasificar los tweets en función de su sentimiento. En este caso, se quiere que se prueben diferentes configuraciones. Se pueden añadir más capas a la red, capas de convolución, capas densas, etc. Se pueden añadir capas con el método `add`. Se trata de experimentar con diferentes arquitecturas cambiando el número y tipo de capas o ajustando parámetros como el número de neuronas por capa o las funciones de activación.

Se proporciona también un código para pintar las curvas de aprendizaje. Estas curvas son gráficos que nos permiten ver cómo el modelo aprende a lo largo de su entrenamiento. En particular, muestran sus resultados al ser evaluados en cada epoch sobre su conjunto de entrenamiento y el conjunto de validación. Se pueden mostrar gráficos distintos para el accuracy y para el error (su comportamiento será opuesto).

Estas curvas se utilizan para diagnosticar si un modelo no es capaz de resolver la tarea, puede sufrir sobre-aprendizaje (se ajusta perfectamente al conjunto de entrenamiento, pero no es capaz de resolver el problema en el conjunto de validación), o si el modelo está bien ajustado.

```python
import matplotlib.pyplot as plt

history_dict = history.history
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']
accuracy = history_dict['accuracy']
val_accuracy = history_dict['val_accuracy']

epochs = range(1, len(loss_values) + 1)
fig, ax = plt.subplots(1, 2, figsize=(14, 6))
#
# Curva accuracy vs Epochs
#
ax[0].plot(epochs, accuracy, 'b', label='Training accuracy')
ax[0].plot(epochs, val_accuracy, 'red', label='Validation accuracy')
ax[0].set_title('Training & Validation Accuracy', fontsize=16)
ax[0].set_xlabel('Epochs', fontsize=16)
ax[0].set_ylabel('Accuracy', fontsize=16)
ax[0].legend()
#
# Curva loss vs Epochs
#
ax[1].plot(epochs, loss_values, 'b', label='Training loss')
ax[1].plot(epochs, val_loss_values, 'red', label='Validation loss')
ax[1].set_title('Training & Validation Loss', fontsize=16)
ax[1].set_xlabel('Epochs', fontsize=16)
ax[1].set_ylabel('Loss', fontsize=16)
ax[1].legend()
```

¿Qué ocurre si se prueba a hacer un preprocesamiento básico de los textos antes de generar el vocabulario? Por ejemplo, eliminar stopwords, signos de puntuación y lematizar. ¿Mejoran los resultados?

Ejercicio 3

Lo mismo que en el ejercicio anterior, pero en lugar de inicializar los pesos de forma aleatoria, que se haga precargando embeddings.

Para ello, se puede utilizar el modelo de embeddings de Word2Vec que hemos utilizado en otros ejercicios, el modelo “SBW-vectors-300-min5.txt”. Se puede descargar a partir del siguiente enlace. Nota que es necesario loggearse con una cuenta en Kaggle:

https://www.kaggle.com/datasets/rtatman/pretrained-word-vectors-for-spanish

Cárgalo mediante el siguiente código de ejemplo. Nota que <TU_PATH> debe reemplazarse por la ubicación donde se encuentra el modelo en tu sistema de archivos:

```
model = KeyedVectors.load_word2vec_format('<TU_PATH>SBW-vectors-
        300-min5.txt', binary=False)
```