Universidad Rey Juan Carlos

TEMA 5 – HOJA DE EJERCICIOS I

Se plantean una serie de ejercicios para practicar con Transformers, ejercicios sencillos como prueba de concepto.

Ejercicio 1

Se propone un ejercicio para ver cómo entrenar un modelo encoder de tipo transformer 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).

Se proporciona el código completo, hay que ir revisándolo para entenderlo de forma general, asociando lo que se ve en el código con lo visto en la parte teórica de clase.

Importar librerías y definir datos de ejemplo.

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.layers import Dense, Input, MultiHeadAttention, LayerNormalization, Embedding, GlobalAveragePooling1D
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

sentences = [
    'Estoy un poco harto del día a día, nada mejora',
    'Hoy es un buen día',
    'No se te ve satisfecho con el trabajo',
    'Este paisaje es hermoso y bonito'
]
labels = [0, 1, 0, 1]  # 1: Positivo, 0: Negativo
```

Preprocesamiento del texto.

```
vocab_size = 1000
max_length = 10

tokenizer = Tokenizer(num_words=vocab_size)
tokenizer.fit_on_texts(sentences)
sequences = tokenizer.texts_to_sequences(sentences)
X = pad_sequences(sequences, maxlen=max_length)

X = tf.convert_to_tensor(X, dtype=tf.float32)
labels = tf.convert_to_tensor(labels, dtype=tf.float32)
```

Crear modelo transformer encoder.

```
def create_transformer_classifier(vocab_size, max_length):
    inputs = Input(shape=(max_length,))

    embedding_layer = tf.keras.layers.Embedding(vocab_size, 32)(inputs)

    attention = MultiHeadAttention(num_heads=2, key_dim=32)(embedding_layer, embedding_layer, embedding_layer)

    x = LayerNormalization()(attention + embedding_layer)

    x = tf.keras.layers.GlobalAveragePooling1D()(x)
    outputs = Dense(1, activation='sigmoid')(x)

    return Model(inputs, outputs)


model = create_transformer_classifier(vocab_size, max_length)
model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy']
)

model.summary()
```

Entrenar el modelo.

```
history = model.fit(
    X, labels,
    epochs=10,
    batch_size=2,
    verbose=1
)
```

Evaluar con nuevas frases.

```
test_sentences = [
    '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'
]

test_sequences = tokenizer.texts_to_sequences(test_sentences)
test_padded = pad_sequences(test_sequences, maxlen=max_length)
test_padded = tf.convert_to_tensor(test_padded, dtype=tf.float32)

predictions = model.predict(test_padded)

for sentence, prediction in zip(test_sentences, predictions):
    sentiment = "Positivo" if prediction > 0.5 else "Negativo"
    print(f"Frase: '{sentence}'")
    print(f"Sentimiento: {sentiment} (probabilidad: {prediction[0]:.2f})")
    print()
```

Se pide:

1. Ejecutarlo y ver el resultado.
2. Comparar el resultado con lo que se obtuvo al utilizar redes neuronales en el tema 4 para estas mismas frases y mismo problema.
3. Probar a ver si aumentando el tamaño del conjunto de entrenamiento mejora.
4. Probar a ver si cambiando la configuración del transformer añadiendo nuevas capas mejora.

   Nota: como se puede ver en el código, no se han utilizado embeddings posicionales.

Ejercicio 2

En este ejercicio se quiere seguir creando un transformer básico, centrándonos en la parte del encoder. En este caso, se quiere saber si una noticia es o no falsa. Se utilizará el conjunto de FakeNews que hemos utilizado en ejercicios del tema 4.

Se trata de hacer lo mismo que en el ejercicio anterior, pero leyendo los datos de las noticias del dataset, que habrá que partir en tres partes: primero train y dev y de la parte de dev, una parte para test.

Se piden dos versiones:

a) No utilizar embeddings de posición, como hemos hecho hasta ahora.
b) Utilizar embeddings de posición. Se puede utilizar para ello la clase PositionEmbedding de la librería keras-nlp.

   Para instalar la librería podéis hacer: pip install keras-nlp

Hay que comparar los resultados obtenidos con y sin embeddings posicionales. ¿Qué ha ocurrido? ¿Los resultados de la clasificación son mejores o peores?

Ejercicio 3

En este ejercicio, se propone entrenar un model decoder de tipo transformer para la predicción de nuevos tokens. Para ello, se realizará un entrenamiento autoregresivo a partir del siguiente conjunto de frases:

```
sentences = [
        "El perro corre feliz",
        "El gato salta ágil",
        "La tortuga camina lenta",
        "El caballo trota fuerte",
        "El perro ladra ruidoso",
        "El gato duerme tranquilo",
        "La tortuga nada lenta",
        "El caballo galopa veloz",
        "El perro juega contento",
        "El gato observa curioso",
        "La tortuga descansa pacífica",
        "El caballo relincha bravo",
```

```
    "El perro huele atento",
    "El gato maúlla suave",
    "La tortuga explora cautelosa",
    "El caballo corre elegante"
]
```

1. Configura los siguientes parámetros de entrenamiento:

   - Tamaño del vocabulario: 40 tokens
   - Longitud máxima de secuencias: 10 tokens
   - Número de dimensiones de los embeddings: 32 dimensiones

   Tokeniza las frases de entrada, transformándolas a una lista de IDs de los tokens que los conforman.

2. Para la obtención de etiquetas que faciliten el entrenamiento, define el conjunto de secuencias de entrada (X) como las frases sin su último token. Por el contrario, define las secuencias de salidas o etiquetas (Y) como las frases sin su primer token.

   Es decir, dada la frase original *[w1, w2, w3, w4, w5]:*

   - La secuencia de entrada (X) sería *[w1, w2, w3, w4]*
   - La secuencia de salida (Y) sería *[w2, w3, w4, w5]*

   Una vez obtenidas las anteriores secuencias, aplica padding sobre las mismas con la función pad_sequences(X, maxlen=max_length). Convierte posteriormente las secuencias resultantes en tensores de Tensorflow.

3. Crea y entrena el modelo decoder basándote en el empleado durante el Ejercicio 1. Realiza las siguientes modificaciones:

   - Modifica la capa de MultiHeadAttention para añadir *causal masking*:

     ```
     attention = MultiHeadAttention(num_heads=2,
     key_dim=embedding_dim)(embedding_layer, embedding_layer,
     embedding_layer,use_causal_mask=True)
     ```

   - Después de la capa de normalización y conexión residual, haz la predicción final del nuevo token con una capa densa del tamaño del vocabulario (sin GlobalAveragePooling1D):

```
outputs = Dense(vocab_size)(x)
```

- Cambia la función de pérdida, usando en su lugar la siguiente:

```
tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
```

4. Predice el siguiente token dada la secuencia de palabras "El caballo". Puedes hacerlo a través de la siguiente función, la cual obtiene el siguiente token de mayor probabilidad:

```
def predict_next_token(model, tokenizer, sentence, max_length):

    seq = tokenizer.texts_to_sequences([sentence])[0]
    seq = pad_sequences([seq], maxlen=max_length)

    predictions = model.predict(seq)
    last_pred_logits = predictions[0, -1]
    next_token_id = np.argmax(last_pred_logits)
    return tokenizer.index_word.get(next_token_id, None)
```

5. Dada la secuencia de palabras "El caballo", predice todos los tokens siguientes hasta llegar a None. Prueba con otras secuencias de palabras y observa los resultados.

Ejercicio 4

En este ejercicio, se propone entrenar una arquitectura completa de transformers que incluya tanto un encoder como un decoder para la traducción automática de textos de inglés a español. Para ello, se partirá de los siguientes conjuntos de frases en dichos idiomas:

```
sentences_english = ["hello world", "goodbye world", "hello everyone"]
sentences_spanish = ["hola mundo", "adiós mundo", "hola a todos"]
```

1. Añade los tokens de comienzo de la frase a generar <bos> y fin de la frase generada <eos> a las frases en español. Puedes hacerlo con la siguiente línea de preprocesamiento:

```
sentences_spanish = [
    f"<bos> {sent} <eos>"
    for sent in sentences_spanish
]
```

2. Tokeniza y añade padding en los inputs del encoder, correspondientes al conjunto de frases en inglés. Puedes utilizar el siguiente código de ejemplo:

```
encoder_tokenizer = Tokenizer(filters='', oov_token='<OOV>')
encoder_tokenizer.fit_on_texts(sentences_english)
encoder_seq = encoder_tokenizer.texts_to_sequences(sentences_english)

encoder_vocab_size = len(encoder_tokenizer.word_index) + 1
encoder_max_len = max(len(seq) for seq in encoder_seq)

encoder_inputs = pad_sequences(encoder_seq, maxlen=encoder_max_len, padding='post')
```

3. Tokeniza y añade padding en las secuencias de entrada y de salida de entrenamiento del decoder, correspondientes al conjunto de frases en español. Puedes fijarte en el Ejercicio 3, definiendo el conjunto de secuencias de entrada del entrenamiento del decoder como las frases sin su último token y las secuencias de salida del entrenamiento del decoder como las frases sin su primer token.
4. Construye con Keras y TensorFlow el bloque del encoder. Puedes basarte en el propuesto durante el Ejercicio 2.
5. Construye con Keras y TensorFlow el bloque del decoder. Puedes utilizar el siguiente código de ejemplo:

```
def transformer_decoder(inputs, encoder_output, head_size, num_heads, ff_dim,
dropout=0.1):

        attn1 = MultiHeadAttention(num_heads=num_heads,
        key_dim=head_size)(query=inputs, value=inputs, key=inputs,
        use_causal_mask=True)
        attn1 = Dropout(dropout)(attn1)
        out1 = LayerNormalization(epsilon=1e-6)(inputs + attn1)

        attn2 = MultiHeadAttention(num_heads=num_heads,
        key_dim=head_size)(query=out1, value=encoder_output, key=encoder_output)
        attn2 = Dropout(dropout)(attn2)
        out2 = LayerNormalization(epsilon=1e-6)(out1 + attn2)

        ff = Dense(ff_dim, activation='relu')(out2)
        ff = Dense(out2.shape[-1])(ff)
        ff = Dropout(dropout)(ff)

        return LayerNormalization(epsilon=1e-6)(out2 + ff)
```

6. Construye y compila el modelo que combina el bloque del encoder y el bloque del decoder. Toma como base la compilación desarrollada durante el Ejercicio 3. Nota que en esta ocasión se le debe pasar al modelo de Tensorflow tanto los inputs del encoder

como del decoder. De este modo, puedes instanciar al modelo utilizando el siguiente código de ejemplo, donde en_in y dec_in son los tensores de entrada del encoder y el decoder (es decir, objetos Input() de Keras):

```
Model([en_in, dec_in], outputs)
```

7. Entrena el modelo compilado. Puede utilizar el siguiente código de ejemplo:

```
model.fit([encoder_inputs, decoder_inputs], decoder_targets, epochs=5)
```

8. Predice la frase a traducir en español dada una frase en inglés. Puedes utilizar el siguiente código de ejemplo, el cual se basa en la predicción de tokens realizada durante el Ejercicio 3:

```
def translate_sentence(input_sentence):
        eos_token = decoder_tokenizer.word_index['<eos>']
        bos_token = decoder_tokenizer.word_index['<bos>']

        encoder_input = encoder_tokenizer.texts_to_sequences([input_sentence])
        encoder_input = pad_sequences(encoder_input, maxlen=encoder_max_len,
        padding='post')

        decoder_output = [bos_token]

        for _ in range(decoder_max_len - 1):
                decoder_input = pad_sequences(
                        [decoder_output],
                        maxlen=decoder_max_len-1,
                        padding='post'
                )
                pred = model.predict([encoder_input, decoder_input], verbose=0)
                next_token = np.argmax(pred[0, len(decoder_output)-1])

                decoder_output.append(next_token)
                if next_token == eos_token:
                        break

        output_tokens = [
                decoder_tokenizer.index_word[token]
                for token in decoder_output
                if token!= eos_token and token!= bos_token
        ]
        return ' '.join(output_tokens)

print(translate_sentence("hello world"))
```